Performance Evaluation of a Class of Neuro-Fuzzy Controllers

A Thesis Submitted

in Partial Fulfillment of the Requirements

for the Degree of

Master of Technology

by

Ratan Bajpai

to the

DEPARTMENT OF ELECTRICAL ENGINEERING

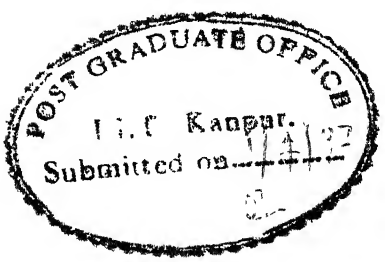
INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

April 1997

EE-1997-M-BAJ-PER

CERTIFICATE

This is to certify that the work contained in the thesis entitled Performance Evaluation of a Class of Neuro-Fuzzy Controllers by Ratan Bajpai has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.



Dr. ARINDAM GHOSH,
Professor,
Department of Electrical Engineering,
Indian Institute of Technology, Kanpur.

Dedicated To

My Grandfather

Acknowledgements

Studying at I.I.T. Kanpur has been one of the very good experiences of my life. As I reflect on my days passed here, I can remember numerous faces who have helped me to evolve into a (hopefully) better person.

Getting through the course work in the first two semesters was a difficult task, but the philosophical interludes of Dr. P.R.K. Rao during his lectures was a great inspirational source. Also, the company of my classmates Peri, Umesh, Nimesh, Sanjay, Gautam, Marathe, Piyush, Sricharan, Manoj and others lessened the burden and made things more enjoyable.

The final semester consisted of the thesis work done under the guidance of Dr. Arindam Ghosh. I gratefully acknowledge the help and freedom of work given by him. The lighter moments shared with him will be remembered for a long time to come.

Being a localite was a great advantage indeed. I never felt the anguish of eating in the Hall IV mess. My sincere gratitudes are towards my family members and relatives, without their support it would have been difficult to sustain through the life at I.I.T. K.

Harmonisation of piano and guitar with Alok was a blissful experience. Together, we also ventured through the various natural surroundings here. Playing basketball and T.T. and watching movies with Suyog, Vikas, Milind, Arvind and Kapil can never be forgotten. The company of my neighbours Karthik and Jagan was also a

precious one. My discussions with Prince on Space-Time and fundamental laws of nature sometimes took us beyond space and time.

Finally, I appreciate the efforts (intentionally or unintentionally) of some of the residents of Hall VI for brightening me up and making me fresh in certain dull moments here. All this and much more makes it difficult to leave this place, but as time flows, I will only be left with the treasure of these wonderful memories.

Abstract

Conventional control techniques do not sufficiently address the problem of control design for complex nonlinear systems. The field of intelligent control was developed for solving such problems by incorporating logic, reasoning and heuristics into the more algorithmic approach provided by conventional control theory. Fuzzy control and neural networks have become two important streams of intelligent control. Various approaches have been proposed in the literature to combine fuzzy and neural systems in order to utilize their synergetic relationship and to compliment each other thereby eliminating their respective drawbacks. This thesis attempts to evaluate the performance of a class of such neuro-fuzzy controllers. The fuzzy-neural system (ANFIS) studied here results on translating a fuzzy inference system into a connectionist network form. This controller is then applied to a simple linear system and two nonlinear control problems. In the first example (*speed control of a permanent magnet dc motor*), the system attempts to track a specified speed trajectory. In the second example (*disturbance rejection in a buck-boost converter system*), the controller tries to make the average output voltage constant when some random disturbance is present in the input dc voltage. The simulation results reveal that the ANFIS controller performed well under certain specific conditions. For its general applicability to real control situations, more theoretical results are needed and some important problems are required to be addressed.

Contents

List of Figures

Chapter 1

Introduction

Many of the physical systems encountered in real world are nonlinear in nature. Their linearity property is exhibited only in a limited operating range i.e. when the physical system does not deviate too much from the *nominal* set of operating conditions. Hence appropriate theoretical tools are required for analysis and synthesis of such systems. Compared with the significant advances in linear control theory, the advances in nonlinear control have been limited. Universally applicable tools and methodologies are not available for such systems and so they have to be dealt on system by system basis.

1.1 Past Developments

The local linearization-based analysis is probably the most widely used analysis and design method for nonlinear control systems. The initial applications of this method involved calculating a linear dynamic model to approximate the nonlinear system at its nominal operating point. Mild nonlinearities could be tolerated, since as long as the system remained near this operating point, the feedback made system insensitive to small dynamic mismodelling. The obvious deficiency of this technique was that significant nonlinearities could not be tolerated, since the linear approximation would be valid only in a small neighbourhood of the nominal operating point.

This simple linear control technique was extended to systems with multiple

significant nonlinearities using gain scheduling. In gain scheduling, the nonlinear system is approximated by performing multiple local linearizations about a variety of trajectories. Once this collection of linear models is established, a linear controller is designed for each of these linear models, and then these controllers are interpolated as the system moves from one trajectory neighbourhood to another. Again this allows the use of linear control design and analysis tools but requires the use of enough approximate linear models to capture the nonlinear dynamics of the plant adequately. In general, the neighbourhood where the approximation is valid may be small, thus requiring a large number of approximate linear models to characterize adequately the nonlinear system truly in its useful operating range. The above requires a lot of effort on the part of control system designer.

The gain scheduling technique essentially approximates the nonlinear system with a time varying linear model. The approximation of nonlinear system with a linear time-varying model inspired the application of adaptive control. In adaptive control, realtime measurements of the plant inputs and outputs are used either to derive explicitly a model for the plant and then design a controller based on this model (indirect adaptive control) or to directly modify the controller (direct adaptive control). Although the principle behind adaptive control is straight forward, the analysis of adaptive control is complicated by the nonlinear and time varying nature of the system. Another thing observed was that the interaction of disturbances and external command signals with even slight mismodelling could lead to severe instability (Rohrs et al. 1980). The reason for this kind of behaviour was that the initial theory did not adequately account for all of the non-ideal conditions present in the realistic control environment.

The current practice of adaptive control comprises the identification of specific parameters. For indirect adaptive control a set of plant parameters is identified, while for direct adaptive control it is a set of controller parameters that is identified. For linear (possible time-varying) plants these parameters are frequently the transfer function coefficients although if there is adequate *a priori* information, physical parameters of the plant or explicit gain in the controller may be identified. The most successful applications of adaptive control are those in which there is enough

a priori information to reduce the number of identified parameters to a manageable set and thereby decrease the computation time substantially. Also important in adaptive control is the fundamental conflict of regulation and identification. For good parameter identification, signals with rich frequency content are desired so that all of the significant dynamics of the system will be identified. However, this is in direct conflict with the goal of regulation. Under good regulation the system is relatively quiescent and the input and output signals will not adequately excite the system, making identification difficult. In control practice this has led to various schemes that turn off the identification when regulation is good, and thus performing both the operations satisfactorily.

Other practical methods for nonlinear control are also available in the literature, but mostly these are applicable to specific systems which are quite simple in nature.

1.2 Intelligent Control

To circumvent the problems mentioned above and to realise more flexible control systems a field of intelligent control was developed which incorporated elements such as logic, reasoning and heuristics into the more algorithmic techniques provided by conventional control theory.

Intelligent control systems reflect the ability to emulate human capabilities, such as planning learning and adaptation Such systems tackle the problem of environmental uncertainty by human-like decision making, heuristic reasoning and learning from past experience. Learning is required when the complexity of a problem or the uncertainty thereof prevents *a priori* specification of a satisfactory solution. Such solutions are then only possible through accumulating information about the problem and using this information to dynamically generate an acceptable solution. Such systems are required to exhibit satisfactory closed loop system performance and integrity over a wide range of operating conditions. The characteristic of the system must therefore relate to the complexity of the plant, including nonlinear and time variant plant behaviour, dimensionality and other multivariable characteristics, the complexity of the desired performance objective, imperfection and uncertainties

in the measurements, and an ability to cope with component failures.

In recent years the use of the terminology *intelligent control* has come to embrace diverse methodologies combining conventional control theory and emergent techniques based on physiological metaphors, such as neural networks, fuzzy logic, artificial intelligence, genetic algorithms and a variety of search and optimisation techniques Amongst these, fuzzy logic and neural network techniques have acquired important positions.

1.2.1 Fuzzy Control

One of the most widely publicised techniques for embodying human-like thinking into a control system is fuzzy control [1]. A fuzzy controller can be designed to roughly emulate the human deductive process. A typical fuzzy controller, as shown in Figure 1.1 consists of four main components: a rule base, a fuzzy inference system, an input fuzzification interface, and an output de-fuzzification interface. The rule

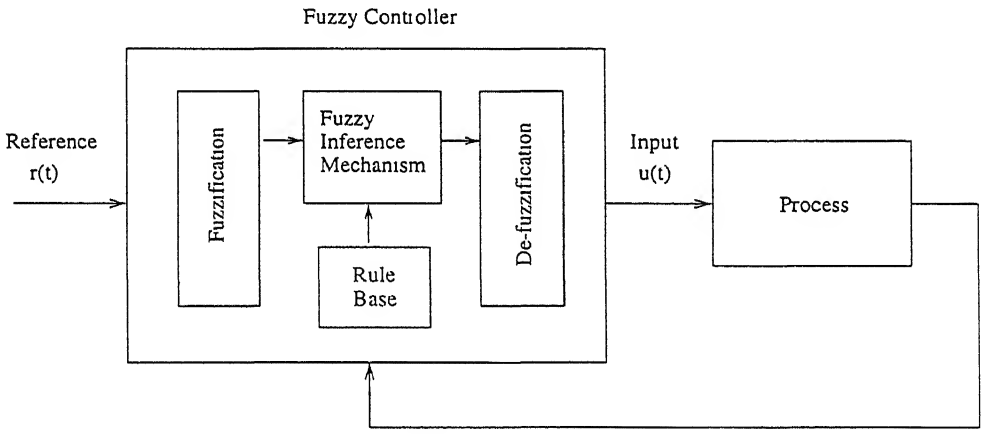


Figure 1.1: Fuzzy Controller

base holds a set of IF - THEN rules that quantify the knowledge that human experts have amassed about solving particular problems. It acts as a resource to the fuzzy inference system, which makes successive decisions about which rules are most relevant to the current situation and applies the actions indicated by those rules. The input fuzzifier takes the *crisp* numeric inputs to the system and, as its name implies, converts them into fuzzy form needed by the fuzzy inference system. At

the output, the de-fuzzification interface combines the conclusions reached by the fuzzy inference system and converts them into crisp numeric values as an output. A typical fuzzy control rule can be of the following type:

> IF $e(t)$ is positive-small AND $e(t)$ is positive-medium,
> THEN $u(t)$ is positive-small.

where $e(t)$ and $\dot{e}(t)$ are error (desired output - actual output) and derivative of error respectively, and $u(t)$ is the control force. *Positive-small* and *positive-medium* are classes into which the crisp input values are put by the fuzzification interface. A complete rule base consists of a whole set of such rules in which linguistic descriptions like *positive medium* etc. are given precise meaning by fuzzy logic. Different models of fuzzy reasoning mechanisms have been developed in the literature, some of which will be described in Chapter 2.

Fuzzy logic control (FLC) has proven effective for complex and imprecisely defined processes for which standard model based control techniques are difficult to apply. There are, however, a number of problems associated with FLC design. A common bottleneck is that the derivation of fuzzy control rules is often time consuming and difficult and relies to a great extent on so called process experts who may not be able to transcribe their knowledge into the requisite rule form. There also exists no formal framework for the choice of parameters of fuzzy systems and hence the means of tuning them, and learning models in general has become an important subject in fuzzy control. Problems with high dimensionality are also often encountered in implementation of fuzzy control systems for multivariable processes, since the number of rules increase rapidly for multiple input systems. Such problems have generated considerable scepticism about what fuzzy control can or cannot achieve, fuelled not least by an absence of fuzzy theoretic formalism. The most common issues of criticism arise from the fact that fuzzy design is still generally a heuristic methodology for which no analytical tools exist to verify or validate the controller performance or stability. Presently however, a number of studies related to analytical issues in fuzzy systems, such as fuzzy modelling and stability analysis,

are being undertaken [2], [3]

1.2.2 Neural Networks for Control

Next to fuzzy logic systems, probably no other intelligent control area has stirred as much interest as the application of artificial neural networks (ANN) for control [4]. The basic design principles used in neural network control (NNC) are not entirely unique to neural network design, but can be viewed as a subset and extension of well known principles from control theory such as dynamic programming.

Neural computation represents a technique by which knowledge is acquired from sets of training examples and stored in a distributed manner in connectionist structure of the network. The distributivity contributes to increased learning capabilities of neural networks because the individual elements in the network are capable of adjusting their connections to achieve near optimal input-output mappings. Also distributive learning permits a response to a novel situation which is inferred using knowledge of previously learned, similar but not exactly same, circumstances. Thus the major advantage of neural network learning is the ability to accomodate poorly modelled, nonlinear dynamical systems.

Neural networks can be classified into two main categories based on their connection structures: feedforward and recurrent [5] (see Figure 1.2). Recurrent (feedback) networks are more suitable for control of dynamical systems, where current outputs depend on past inputs and outputs. The output of a feedforward network is a function only of its current inputs and hence cannot truly model dynamic behaviour. But feedforward network are the most commonly used type, mainly because of difficulty of training and stability in feedback networks. Feedforword networks have been successfully applied in identification and control of dynamic systems by explicitly using the required number of past inputs and outputs in the training procedure [6], [7].

Various approaches of neural network application to control problems are available in the literature. Some of major methods include:

1. Supervised control [18].

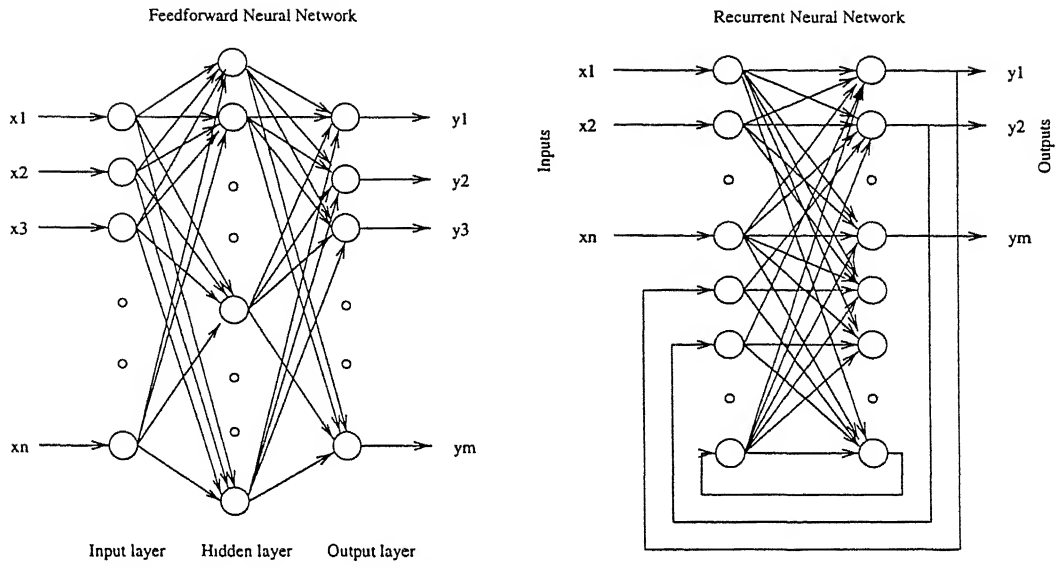


Figure 1.2: Feedforward and Recurrent networks

2. Direct inverse control [20].

3. Direct adaptive control [20].

4. Indirect adaptive control [23], [24].

5. Model reference control [18].

These methods will be discussed in detail in Chapter 3. The drawback of neural systems is the difficulty in interpreting the knowledge stored in the network. The distributed nature of such systems does not provide a strong scheme of knowledge representation. To overcome this hurdle methods have been proposed to integrate fuzzy logic with neural networks and this resulted in the development of Fuzzy-Neural Systems.

1.2.3 Synthesis of Fuzzy and Neural Systems

Fuzzy logic and neural systems have contrasting control application requirements. On one hand, fuzzy control systems are appropriate if sufficient expert knowledge about the process is available, while on the other, neural systems are useful if sufficient process data is available or measurable. Despite this there exists a lot

lot of similarity and a synergetic relationship between neural networks and fuzzy logic systems [8]. Formal equivalences between different types of fuzzy and neural systems have also been established [9], and it has been shown that neural networks can be constructed that are identical to fuzzy logic controllers [10]. Both approaches build nonlinear systems based on bounded continuous variables, the difference being that neural systems are treated in a numeric (quantitative) manner whereas, fuzzy systems are treated in a symbolic (qualitative) manner. By combining the two things we can use the strength of each one in reinforcement of the other in the resulting fuzzy-neural system In particular, neural networks do not provide a strong scheme for knowledge representation while fuzzy logic controllers, in their basic form at least, do not have capabilities of automated learning. A fuzzy-neural system would posses a representational scheme that can portrait knowledge in an explicit manner by providing excellent reasoning interface and understandable model structure having both knowledge acquisition and learning capability.

1.3 Objective and Organisation of the Thesis

The objective of this thesis is to evaluate the performance of a class of neuro-fuzzy controllers These controllers are derived based on a fuzzy-neural system known as *Adaptive Network Based Fuzzy Inference System* (ANFIS). Chapter 2 of the thesis describes ANFIS architecture and its learning algorithm Chapter 3 discusses various control structures that can be derived using ANFIS. In this chapter the performance of this class of controllers is observed using a simple linear model. To investigate further, two case studies are presented in Chapters 4 and 5. In Chapter 4, the results of this controller, when used for the speed regulation of a permanent magnet dc motor are presented, while those for the disturbance rejection in a buck-boost converter system are presented in Chapter 5. The thesis concludes with suggestions for further work in Chapter 6.

Chapter 2

Adaptive Network based Fuzzy Inference System (ANFIS)

Fuzzy and neural systems have been integrated at different levels and in various ways to give several fuzzy-neural paradigms, yet two distinctive approaches of synthesis can be identified. On one hand, many paradigms use multilayered feedforward neural networks to approximate a fuzzy control algorithm [11]. On the other, there are approaches which translate a fuzzy inference system to a connectionist network form and then tune the parameters of the fuzzy system [12]. The fuzzy-neural system used here (ANFIS) belongs to the latter class. It converts a fuzzy model, properly known as the Sugeno fuzzy model [13], into a network form and tunes parameters of the model for near optimal input output mappings.

2.1 Fuzzy Inference System

As indicated earlier a fuzzy controller has four main components: a rule base, a fuzzy inference system, an input fuzzification interface and an output defuzzification interface Sometimes the rule base is considered as a part of fuzzy inference system. The decision making logic in the fuzzy logic controller is handled by the fuzzy inference system and hence it is also known as the kernel of fuzzy controller. It makes the choice of rule to be fired from the rule base in the current input conditions.

Fuzzy inference system is based on the concepts of fuzzy set theory, fuzzy if-then rules and fuzzy reasoning. Thus for understanding its functionality some concepts of fuzzy if-then rules and fuzzy reasoning will have to be reviewed.

2.1.1 Fuzzy *If-Then* rules

A fuzzy if-then rule (fuzzy implication or fuzzy conditional statement) takes the following form:

$$\text{if } x \text{ is } A \text{ then } y \text{ is } B \qquad \text{(also written as } A \rightarrow B\text{)}$$

where x and y are fuzzy variables and A and B are linguistic values defined by fuzzy sets and can be written as:

$$\begin{aligned} A &= \{(x, \mu_A(x)) \mid x \in X\} \qquad \text{and} \\ B &= \{(y, \mu_B(y)) \mid y \in Y\} \end{aligned}$$

where X and Y are universe of discourses and $\mu_A(x)$ and $\mu_B(y)$ are associated membership functions (MFs). A realistic example of a fuzzy rule is:

$$\text{if pressure is } \textit{high} \text{ then volume is } \textit{small}$$

here pressure and volume are fuzzy variables and *high* and *small* are defined by appropriate fuzzy sets formed by associating a membership function to the fuzzy variables. The fuzzy rule above, in essence, describes a relation between two fuzzy variables x and y, and can be defined as a binary fuzzy relation R on the product space $X \times Y$. Alternatively a binary fuzzy relation can be viewed as a fuzzy set with universe $X \times Y$, characterised by a two dimensional MF $\mu_R(x, y)$.

2.1.2 Fuzzy Reasoning

Fuzzy reasoning (also known as approximate reasoning) is an inference procedure used to derive conclusions from a set of fuzzy if-then rules and one or more conditions. The fuzzy reasoning mechanism can be illustrated with the help of following example:

Let x and y be two fuzzy variables and A, A' & B, B' linguistic values (fuzzy sets) on X, X, Y and Y respectively. Using the truth of proposition A and fuzzy relation $A \rightarrow B$ on $X \times Y$ we can infer the truth of proposition B as follows:

$$\begin{array}{ll} \text{premise 1 (fact) :} & x \text{ is } A', \\ \underline{\text{premise 2 (rule) :}} & \underline{\text{if } x \text{ is } A \text{ then } y \text{ is } B,} \\ \text{consequence (conclusion) :} & y \text{ is } B'. \end{array}$$

For instance if we have the implication rule *if tomato is red then it is ripe* $(A \rightarrow B)$ and we know that *the tomato is more or less red* $(x \text{ is } A')$ then we may infer that *the tomato is more or less ripe* $(y \text{ is } B')$. This inference procedure is known as generalized modus ponens. Here the fuzzy set B' can be formed by using sup-star compositional rule of inference [14] and is given by:

$$\begin{aligned} \mu_{B'}(y) &= max_x \; min \; [\mu_{A'}(x), \mu_R(x,y)] \\ &= \vee_x \; [\mu_{A'}(x) \wedge \mu_R(x,y)] \end{aligned} \tag{2 1}$$

where $\vee$ indicates union and $\wedge$ intersection. In the above equation we have used max-min composition, we can also use max-product composition i.e. replace intersection with algebraic product, Equivalently,

$$B' = A' \circ R = A' \circ (A \rightarrow B) \tag{2.2}$$

equation (2 1) can be further simplified as [14]

$$\begin{aligned} \mu_{B'}(y) &= \vee_x[\mu_{A'}(x) \wedge \{\mu_A(x) \wedge \mu_B(y)\}] \\ &= [\vee_x\{\mu_{A'}(x) \wedge \mu_A(x)\}] \wedge \mu_B(y) \\ &= w \wedge \mu_B(y) \end{aligned} \tag{2.3}$$

here w is known as firing strength of the fuzzy rule $A \rightarrow B$. Thus w is found as maximum of $\mu_{A'}(x) \wedge \mu_A(x)$, shown in Figure 2.1; where the MF of resulting B' is equal to MF of B clipped at w, shown as shaded area in the consequent part of Figure 2.1

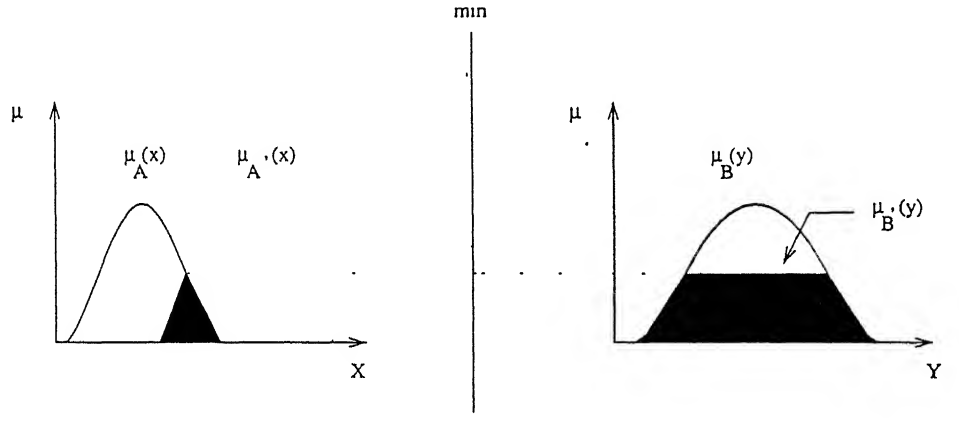


Figure 2.1. Fuzzy reasoning for a single rule with a single antecedent

In the above example we had single rule with only one antecedent (premise variable). We can also have multiple rules with multiple antecedents and they can be analysed in the same fashion

2.2 Different Fuzzy Inference Models

After going through the basic fuzzy reasoning mechanism we can now discuss some of the fuzzy inference models proposed in literature.

2.2.1 Mamdani's Fuzzy Model

Mamdani's fuzzy model [15] was initially used to control a steam engine and boiler combination by obtaining a set of control rules from experienced human operators. This model can be illustrated by the following example having two rules and each rule consisting of two antecedents (premise variables).

premise 1 (fact) :	x is A' and y is B',
premise 2 (rule 1) :	if x is A_1 and y is B_1 then z is C_1,
premise 3 (rule 2) :	if x is A_2 and y is B_2 then z is C_2,
consequence (conclusion) :	z is C'

To obtain the fuzzy set C', let $R_1 = A_1 \times B_1 \rightarrow C_1$ and $R_2 = A_2 \times B_2 \rightarrow C_2$. Then

$$C' = (A' \times B') \circ (R_1 \cup R_2) \tag{2.4}$$

Since $\circ$ operator is distributive over $\cup$, we have

$$\begin{aligned} C' &= [(A' \times B') \circ R_1] \cup [(A' \times B') \circ R_2] \\ &= C'_1 \cup C'_2 \end{aligned} \tag{2.5}$$

where C'_1 & C'_2 are given by [14]

$$\left.\begin{aligned} C'_1 &= w_1 \wedge w_2 \wedge \mu_{c_1}(z) \\ C'_2 &= w_3 \wedge w_4 \wedge \mu_{c_2}(z) \end{aligned}\right\} \tag{2.6}$$

and

$$\left.\begin{aligned} w_{1,3} &= \vee_x \left[\mu_{A'}(x) \wedge \mu_{A_{1,2}}(x)\right] \\ w_{2,4} &= \vee_y \left[\mu_{B'}(y) \wedge \mu_{B_{1,2}}(y)\right] \end{aligned}\right\} \tag{2.7}$$

This reasoning procedure is shown in Figure 2.2 using max and min operators in place of fuzzy *AND* & *OR*. The crisp output can be found by defuzzifying the fuzzy set C'. Various defuzzification strategies such as centroid of area, bisector of area, mean of maximum etc. can be employed for this purpose.

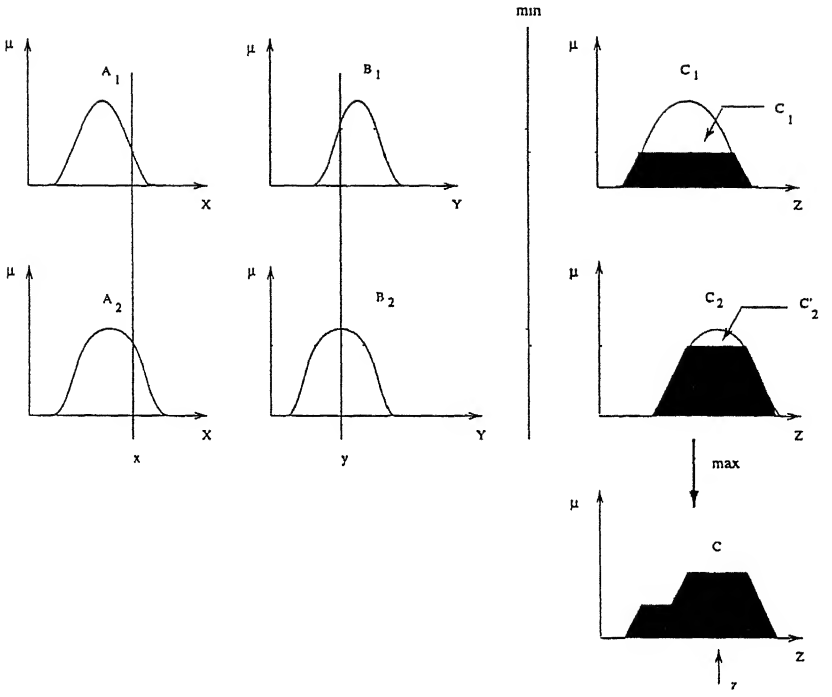


Figure 2.2: Mamdani's fuzzy inference system

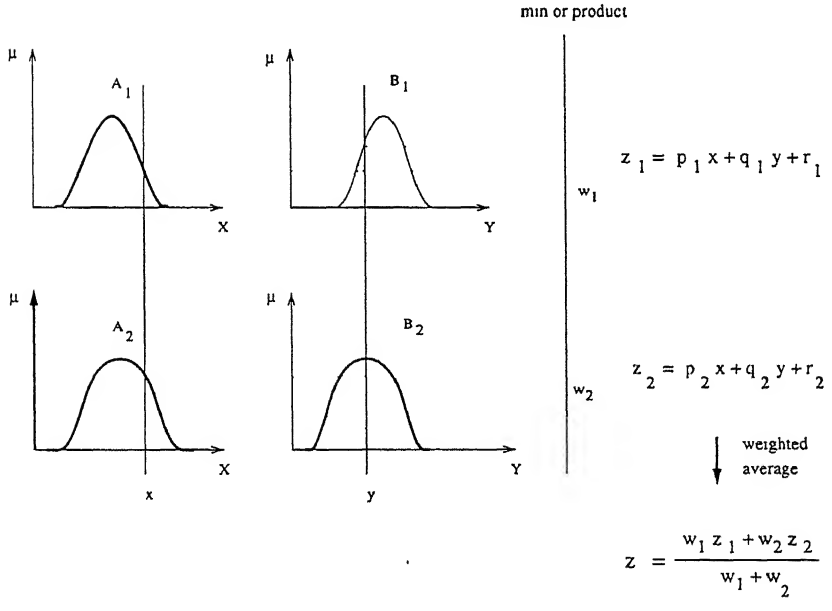


Figure 2.3: Sugeno fuzzy inference system

2.2.2 Sugeno Fuzzy Model

This reasoning mechanism was proposed by Takagi and Sugeno [13], in order to develop a fuzzy model of a system using input-output data set. Rules of the following form are used in this model

$$\text{if } x \text{ is } A \text{ and } y \text{ is } B \text{ then } z = f(x, y)$$

where A & B are fuzzy sets in the antecedent and $z = f(x, y)$ is a crisp function in the consequent. Here the partitioning of the input space is done by the antecedent and the output in the resulting regions is defined by the function f. Usually $f(x, y)$ is taken as a linear function in x & y, but it can be any function as long as it can appropriately describe the output of the system within the fuzzy region specified by the antecedent of the rule. When $f(x, y)$ is a first order polynomial, the resulting fuzzy system is called as first-order Sugeno fuzzy model. When f is a constant, we then have a zero-order Sugeno fuzzy model, which can be viewed as a special case of Mamdani's fuzzy inference system, in which each rule's consequent is specified by a fuzzy singleton (or a pre-defuzzified consequent). Also, a zero order Sugeno fuzzy model is functionally equivalent to a radial basis function network under certain minor constraints [9]. Figure 2.3 shows the fuzzy reasoning procedure for a first-order Sugeno fuzzy model. Since each rule has a crisp output, the overall output is obtained via weighted average and thus the time consuming process of defuzzification is avoided.

2.2.3 Tsukamoto Fuzzy Model

In the Tsukamoto fuzzy model [16], the consequent of each fuzzy if-then rule is represented by a fuzzy set with a monotonical MF, as shown in Figure 2.4. As a result, the inferred output of each rule is defined as a crisp value induced by the rule's firing strength. The overall output is taken as the weighted average of each rule's output. Figure 2.4 illustrates the whole reasoning procedure for a two input, two rule system.

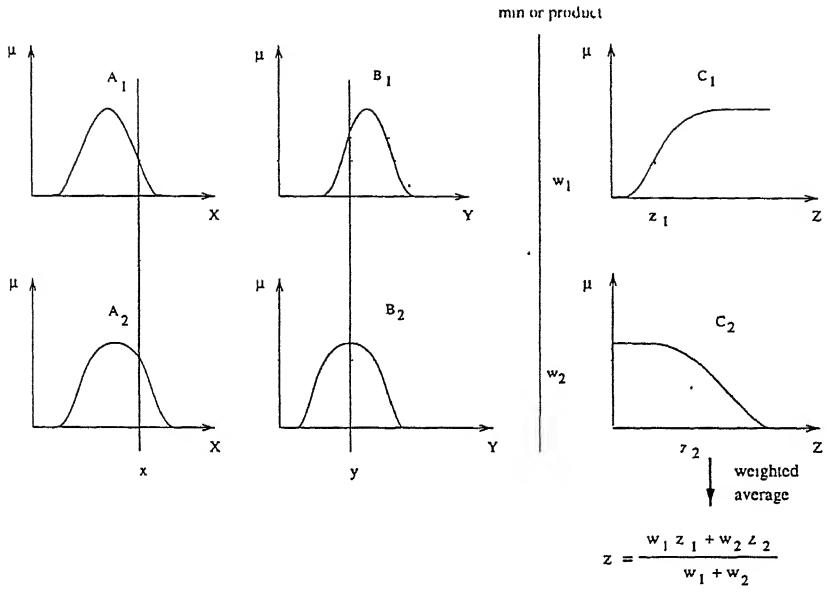


Figure 2.4: Tsukamoto fuzzy inference system

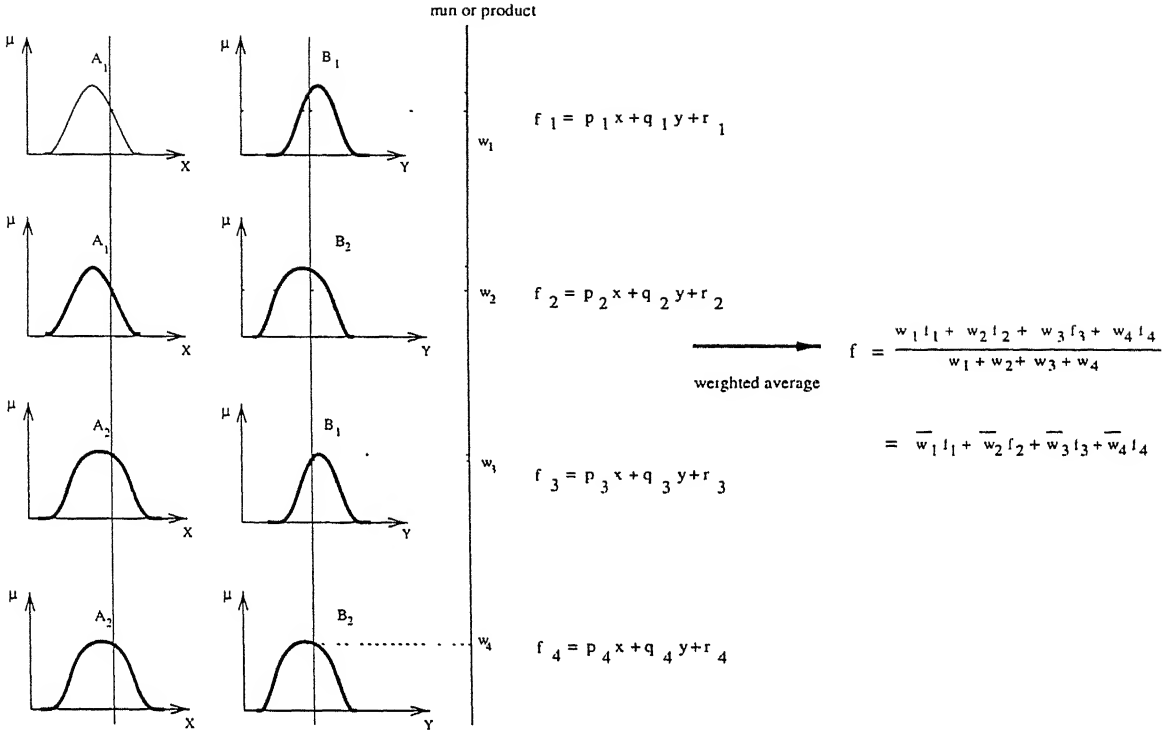
2.3 ANFIS Architecture

The Sugeno fuzzy model discussed above is the inference system which is used to form the adaptive network (ANFIS) [17]. For understanding the structure of ANFIS we take the following example having four Sugeno fuzzy rules, each rule comprising of two antecedents and one consequent linear function. The rules are expressed as :

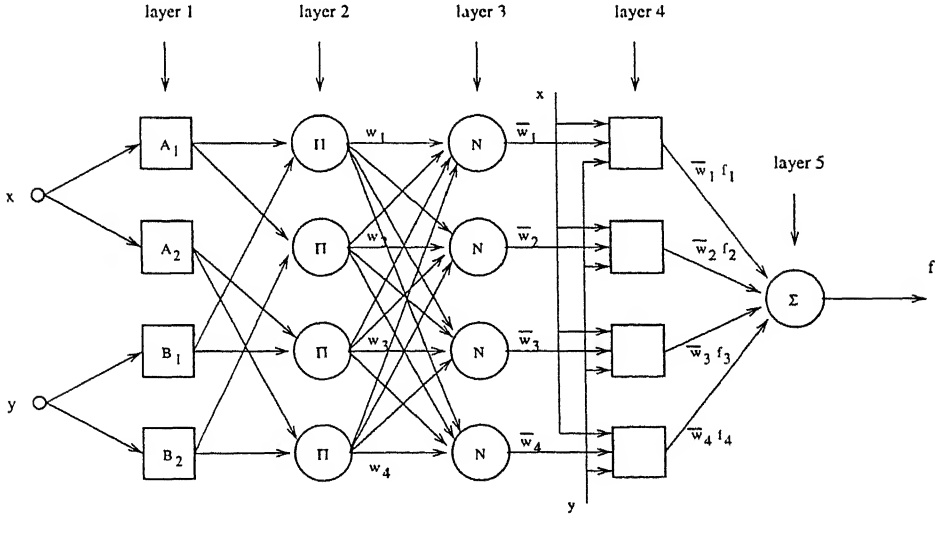
rule 1 : if x is A_1 and y is B_1 then $f_1 = p_1x + q_1y + r_1$.
rule 2 : if x is A_1 and y is B_2 then $f_2 = p_2x + q_2y + r_2$.
rule 3 : if x is A_2 and y is B_1 then $f_3 = p_3x + q_3y + r_3$.
rule 4 : if x is A_2 and y is B_2 then $f_4 = p_4x + q_4y + r_4$.

where each input is associated with two linguistic labels (fuzzy sets). Figure 2.5(a) illustrates the reasoning mechanism for this Sugeno model. The corresponding equivalent ANFIS architecture is shown in Figure 2.5(b). The antecedent part of these four rules partitions the input space into four regions as shown in Figure 2.6, and the output in each region is determined by the consequent part (f_1 to f_4).

The network, which is of feedforward type, consists of five layers, each layer



(a)



(b)

Figure 2.5: (a) A two-input first-order Sugeno fuzzy model with four rules, (b) equivalent ANFIS architecture

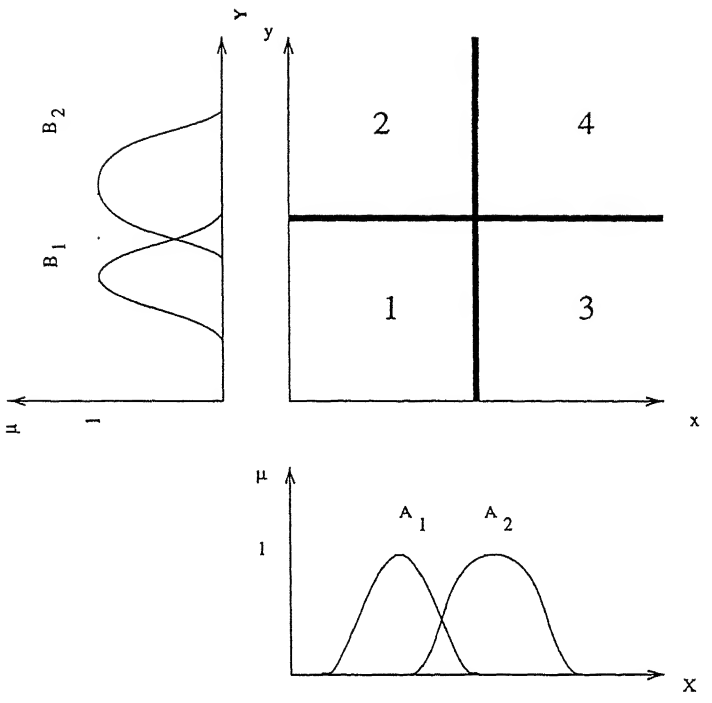


Figure 2.6· Partitioning of input space by the rules in Sugeno fuzzy model

having several nodes. The nodes of preceeding layer are connected to those in the next layer through directional links. There are no links between the nodes of the same layer. Adaptive (modifiable) nodes are shown by squares, while the fixed ones are indicated by circles. Each node is a processing unit that performs a static node function on the incoming signal to generate a single node output and each link specifies the direction of signal flow from one node to another. No weights are associated with the links. Nodes of the same layer performs similar function on the incoming signals. Node function in the adaptive node is a parameterized function with modifiable parameters; by changing these parameters, we are actually changing the node function as well as the overall behaviour of the adaptive network.

The specific description of each layer and its function is given below (we denote the output of node i in layer l as $O_{l,i}$).

layer 1 : This is also called as the fuzzification layer. Crisp values of the input variables are taken as inputs and the nodes give fuzzy values as outputs. Every node i is an adaptive node with a node output defined by

$$O_{1,i} = \mu_{A_i}(x), \qquad \text{for } i = 1,2 \quad \&$$
$$O_{1,i} = \mu_{B_{i-2}}(y), \qquad \text{for } i = 3,4$$

where x, y are the inputs to the nodes and A_i, B_{i-2} are fuzzy sets associated with the nodes. In other words, outputs of this layer are the membership values of the premise part. Here the membership functions for A_i and B_i are the generalized bell functions of the form:

$$\mu_{A_i}(x) = \frac{1}{1 + \left[\left(\frac{x-c_i}{a_i}\right)^2\right]^{b_i}}$$

where $\{a_i, b_i, c_i\}$ is the parameter set of a particular node i. Parameters in this layer are referred to as premise parameters. The number of membership functions per input variable indicates the number of categories into which the input variable is divided or fuzzified by this layer. In this example here, the input variable x is divided into two linguistic labels A_1 and A_2 (see Figure 2.5(b)) having associated MFs $\mu_{A_1}(x)$ & $\mu_{A_2}(x)$ respectively.

layer 2 : Every node in this layer is a fixed (non-modifiable) node labeled II, which multiplies the incoming signals and outputs the product. The antecedent part (*if* part) of rules of the inference system are formed in this layer or in other words, nodes in this layer compute the firing strengths of different rules in the fuzzy reasoning mechanism. The outputs in this layer (as shown in Figure 2.5(b)) are given by

$$O_{2,1} = w_1 = \mu_{A_1}(x) \times \mu_{B_1}(y)$$

$$O_{2,2} = w_2 = \mu_{A_1}(x) \times \mu_{B_2}(y)$$

$$O_{2,3} = w_3 = \mu_{A_2}(x) \times \mu_{B_1}(y)$$

$$O_{2,4} = w_4 = \mu_{A_2}(x) \times \mu_{B_2}(y)$$

To illustrate the rule formation we can see that the antecedent part of rule 1 of the model described above (i.e. if x is A_1 and y is B_1) is formed by node 1, using product for fuzzy *AND* operator. Antecedents of other three rules is also formed in the same fashion by subsequent nodes in the layer.

layer 3 : Every node in this layer is a fixed node labeled N. The nodes in this layer compute the normalized firing strengths of each rule i.e. the ith node calculates the ratio of the ith rule's firing strength to the sum of all other rule's firing strength.

$$O_{3,i} = \overline{w_i} = \frac{w_i}{\sum w_i}$$

layer 4 : Nodes in this layer are modifiable nodes with the following node function:

$$O_{4,i} = \overline{w_i} f_i = \overline{w_i}(p_i x + q_i y + r_i), \qquad i = 1 \text{ to } 4$$

where $\overline{w_i}$ is the output of layer 3 and $\{p_i, q_i, r_i\}$ is the parameter set of node i. The consequent part (*then* part) of the rule is formed in this layer, hence the parameters $\{p_i, q_i, r_i\}$ are referred to as consequent parameters.

layer 5 : The single node in this layer is a fixed node labeled Σ, which calculates the overall ANFIS output as the summation of all incoming signals:

$$O_{5,1} = \sum_i \overline{w_i} f_i$$

The functioning of the consequent part of the network can be explained as follows: If rule 1 gets fired i e. its normalized firing strength ($\overline{w_1}$) is close to unity, then the

component $\overline{w_1} f_1$ will dominate all other parts in the final network output f. Hence the output of the network will be defined largely by $f_1 = p_1 x + q_1 y + r_1$.

Clearly the network described above functions in the same way as first order Sugeno fuzzy model. Note that the fuzzification part is also taken care by ANFIS and since Sugeno fuzzy model gives crisp output, no defuzzification is required here.

2.4 ANFIS Learning Algorithm

Layer 1 and layer 4 in the network contain nodes which are modifiable, having their own (local) parameter set. A union of all the parameters of such nodes forms the network's overall parameter set which controls the static mapping formed by the network. Desired nonlinear mapping can be obtained by tuning the parameters using a training data set consisting of a number of desired input-output pairs of a target system. This tuning procedure is also known as network learning. In the coming part we will discuss two learning algorithms: gradient descent learning and hybrid learning [17].

2.4.1 Gradient Descent Learning

The gradient descent learning algorithm, more commonly known as the back propagation algorithm, is the most well known and widely used training method among the current types of feedforward adaptive networks available. The central part of this learning rule concerns how to recursively obtain a gradient vector in which each element is defined as the derivative of an error measure with respect to a parameter. This is done by means of a chain rule and the gradient vector is calculated by propagating the errors in the direction opposite to the flow of output of each node. Details of the algorithm are described below.

Suppose the given adaptive network has l layers and the kth layer has $\#(k)$ nodes. We denote the output and node function of ith node in kth layer by f_i^k and O_i^k respectively. This output O_i^k depends on the incoming signals to the node and its parameter set. So we have:

$$O_i^k = f_i^k (O_1^{k-1}, \ldots O_{\#(k-1)}^{k-1}, a_i^k, b_i^k, \ldots) \tag{2.8}$$

where a_i^k, b_i^k etc are the parameters of the node. Assuming that given data set has P entries we can define an error measure (or energy function) for the pth $(1 \leq p \leq P)$ entry of training data set as the sum of squared errors

$$E_p = \sum_{m=1}^{\#(l)} (T_{m,p} - O_{m,p}^l)^2 \tag{2.9}$$

where $T_{m,p}$ is the mth component of pth target output vector, and $O_{m,p}^l$ is the mth component of actual output vector produced by presenting pth input vector to the network. Thus the overall error measure is given by:

$$E = \sum_{p=1}^{P} E_p \tag{2.10}$$

For developing a learning procedure that implements gradient descent in E over the parameter space, we will have to calculate the *error rate* $\partial E_p / \partial O$ for pth training data sample and for each node output O. The error rate for the ith node in the output layer l can be calculated using

$$\frac{\partial E_p}{\partial O_{i,p}^l} = -2(T_{i,p} - O_{i,p}^l) \tag{2.11}$$

For the internal node i in layer k, the error rate can be derived using the chain rule:

$$\frac{\partial E_p}{\partial O_{i,p}^k} = \sum_{m=1}^{\#(k+1)} \frac{\partial E_p}{\partial O_{m,p}^{k+1}} \frac{\partial O_{m,p}^{k+1}}{\partial O_{i,p}^k} \tag{2.12}$$

where $1 \leq k \leq l-1$. That is, the error rate of an internal node can be expressed as a linear combination of the error rates of the nodes in the next layer. Therefore

for all the nodes in the network i.e. for $1 \leq k \leq l$ and $1 \leq i \leq \#(k)$, we can find $\partial E_p / \partial O_{i,p}^k$ using equations (2.14) and (2.15).

Now if α is a parameter of the given adaptive network, we have

$$\frac{\partial E_p}{\partial \alpha} = \sum_{O^* \in S} \frac{\partial E_p}{\partial O^*} \frac{\partial O^*}{\partial \alpha} \tag{2.13}$$

where S is the set of nodes whose output depends on α. Then the derivative of the overall error measure E with respect to α is

$$\frac{\partial E}{\partial \alpha} = \sum_{p=1}^{P} \frac{\partial E_p}{\partial \alpha} \tag{2.14}$$

Accordingly, the update formula for the generic parameter α is given by

$$\triangle \alpha = -\eta \frac{\partial E}{\partial \alpha} \tag{2.15}$$

in which η is called as the learning rate which can be further expressed as

$$\eta = \frac{k}{\sqrt{\sum_\alpha \left(\frac{\partial E}{\partial \alpha}\right)^2}} \tag{2.16}$$

where k is the *step-size* which determines the length of each gradient transition in the parameter space. Usually, we can change the value of k to vary the speed of convergence.

Here we can have two learning paradigms depending on whether we update the network parameters based on E or E_p. If we change the parameter α based on E using equation (2.17) then it is called as batch learning (or off-line learning). In this the update action takes place only after the whole training data set has been presented, i.e., only after each *epoch* or *sweep*. On the other hand, if we want the parameters to be updated immediately after each input-output pair has been

presented, then the update formula is based on E_p using equation (2.16) and it is referred to as pattern learning (or on-line learning).

2.4.2 Hybrid Learning Rule

Although we can apply the gradient method to identify the parameters in the adaptive network, the method is generally slow and likely to become trapped in local minima. Here we discuss a hybrid learning rule [17] which combines the gradient method described above and the least squares estimation (LSE) technique to identify the network parameters

For simplicity, let us assume that the adaptive network has only one output, which is given by:

$$output \;=\; F(\vec{I}, S) \tag{2.17}$$

where $\vec{I}$ is the set of input variables and S is the set of parameters. If there exists a function H such that the composite function $H \circ F$ is linear in some elements of S, then these elements can be identified by the least squares method. More formally, if the parameter set S can be decomposed into two sets

$$S \;=\; S_1 \oplus S_2 \tag{2.18}$$

(where $\oplus$ represents direct sum) such that $H \circ F$ is linear in elements of S_2, then on applying H to equation (2.17), we have

$$H(output) \;=\; H \circ F(\vec{I}, S) \tag{2.19}$$

which is linear in elements of S_2. Now given the values of elements of S_1, we can plug P training data into (2.19) and obtain a matrix equation:

$$A\,X \;=\; B \tag{2.20}$$

where X is an unknown vector whose entries are elements of S_2. If there are M elements in S_2, then the dimensions of A, X and B are $P \times M$, $M \times 1$ and $P \times 1$ respectively. Since P (number of training data pairs) is usually greater than M (number of linear parameters), this is an overdetermined problem and generally there is no exact solution to (2.20). Instead, a *least squares estimate (LSE)* of X, X^* is sought to minimize the squared error $\| AX-B \|^2$. This is a standard problem and most well known formula for X^* uses the pseudo inverse of X:

$$X^* \;=\; (A^TA)^{-1}A^TB \tag{2.21}$$

where A^T is the transpose of A, and $(A^TA)^{-1}A^T$ is the pseudo inverse of A if A^TA is non-singular. The above formula is expensive in computation when dealing with the matrix inverse and, moreover, it becomes ill-defined if A^TA is singular. To avoid this, we employ iterative formulas to compute the LSE of X. Specifically, let the ith row vector of matrix A defined in (2.20) be a_i^T and the ith element of B be b_i^T, then X can be calculated iteratively by using the following sequential formulas:

$$\left.\begin{array}{rcll} X_{i+1} &=& X_i \;+\; C_{i+1}a_{i+1}(b_{i+1}^T \;-\; a_{i+1}^T X_i) & \\ C_{i+1} &=& C_i \;-\; \dfrac{C_i a_{i+1} a_{i+1}^T C_i}{1 \;+\; a_{i+1}^T C_i a_{i+1}} & i = 0,1,\ldots,P-1 \end{array}\right\} \tag{2.22}$$

where C_i is called as the *covariance matrix* and the least square estimate X^* is equal to X_P. The initial conditions to bootstrap (2.22) are $C_0 = \gamma I$ and $X_0 = 0$, where γ is a large positive number and I is the identity matrix of dimension $M \times M$.

In the network ANFIS the parameter set S_1 corresponds to premise parameters (parameters in layer 1) and parameter set S_2 corresponds to the linear consequent parameters (parameters in layer 4). We can combine the gradient descent method

and LSE to update the premise and consequent parameters in ANFIS. Each epoch of this hybrid learning procedure is composed of a forward and a backward pass. In the forward pass, we give input data and the functional signals propagate forward to calculate each node output until the matrices A and B in (2.20) are obtained, and the parameters in S_2 are identified by sequential least squares formulas in (2.22). After identifying parameters in S_2, the functional signals keep going forward till the error measure is calculated. In the backward pass, the error rates (the derivative of error measure w.r t each node output given by (2.11) and (2.12)) propagate from the output end towards the input end, and the parameters in S_1 are updated by the gradient descent method using (2.15). For given fixed values of parameters in S_1, the parameters in S_2 thus identified are optimum in the S_2 parameter space due to the choice of squared error measure. The hybrid learning rule described decreases the dimension of search space and cuts down the convergence time.

Here also we can have two learning paradigms: the batch (off-line) learning and pattern (on-line) learning. For batch learning gradient descent should be based of E (see (2.14)) and for pattern learning equation (2.13) using E_P is applied.

2.5 Practical Considerations While Using ANFIS

In a conventional fuzzy inference system, the number of rules in the rule base are decided by the experts familiar with the system to be modeled. While using ANFIS, this information, if available, can be incorporated in the network structure, otherwise the number of MFs assigned to each input (which determines the number of rules) are chosen by trial and error. This situation is much the same as that in neural networks: the number of nodes in hidden layers is decided by using certain rule of thumb. After the number of MFs associated with each input is fixed, the initial values of premise parameters are set in such a way that the MFs are equally spaced along the operating range of each input variable [17].

The step-size k which occurs in (2.16) influences the speed of convergence of the learning algorithm. If k is small, the gradient method will closely approximate the gradient path, but the convergence will be slow. On the other side, if k is large,

convergence will initially be very fast, but the algorithm will oscillate about the optimum. We can vary k adaptively to give improved performance.

Chapter 3

Control Applications Using ANFIS

The area of nonlinear control systems forms a potential field for application of the adaptive network introduced in Chapter 2. A number of approaches for the control of nonlinear dynamical systems using neural networks are available in the literature. We can utilise these techniques in designing a controller for such systems using ANFIS. This chapter introduces common design methods for ANFIS controllers which are mostly derived by directly replacing neural networks with ANFIS.

3.1 Different Control Structures Using ANFIS

A typical feedback control system (shown in Figure 3.1) consists of a plant and a controller. The plant is usually represented by a set of differential (or difference) equations which describe the dynamics of the physical system to be controlled. The plant's response is indicated by the state vector $x(k)$, which is assumed to be accessible in our discussion. In contrast, the controller is usually a static function g which maps the plant state and desired output $x(k)$, $x_d(k)$ into a control action $u(k)$ so that the combined system achieves a given control objective. Thus for a general feedback control system in discrete time domain, we have:

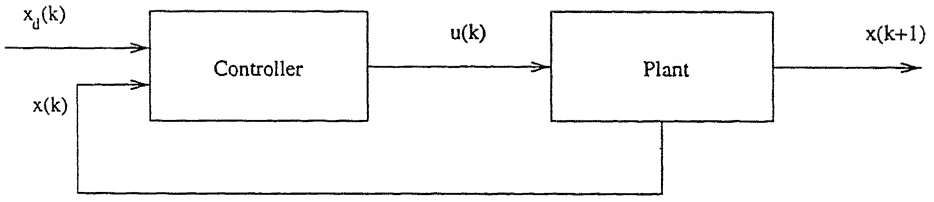


Figure 3.1: Block diagram for discrete time feedback control system

$$\left.\begin{array}{rcll} x(k+1) & = & f[x(k), u(k)] & \text{plant} \\ u(k) & = & g[x(k), x_d(k)] & \text{controller} \end{array}\right\} \tag{3.1}$$

A central problem in control engineering is that of finding the control action u as a function of plant state x in order to achieve a given control goal. Different design methods exist for solving the above mentioned problem and some important techniques are as explained below.

3.1.1 Supervised Control

Supervised control [18] involves training of the adaptive network under supervision of an external agency, which can be a human operator or an expert. This is shown in Figure 3.2. Sometimes a conventional controller is also used for training the network, as shown in Figure 3.3 in which a PI(D) controller provides the required training data. In this method (also known as *feedback error learning* [19]), the performance of ANFIS controller is limited by the original controller. This approach is only advantageous in situations where the original controller is computationally more expensive, which occurs in systems with fast dynamics.

3.1.2 Direct Inverse Control

In direct inverse control (also known as *generalized learning*) the adaptive network is trained to learn the inverse dynamics of the given plant [20]. Then in the actual control phase the network is connected as controller in cascade with the plant. This approach can be explained with the help of Figure 3.4. For a single input single

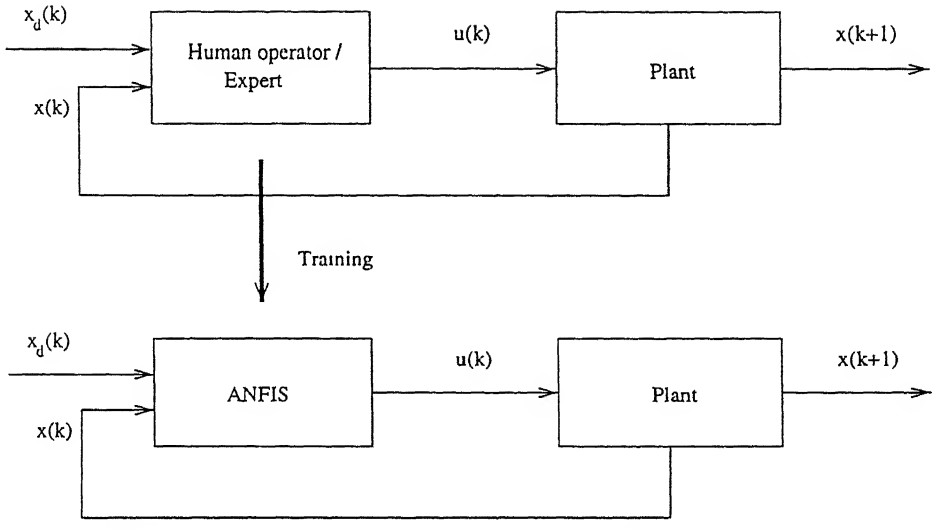


Figure 3.2: Supervised control

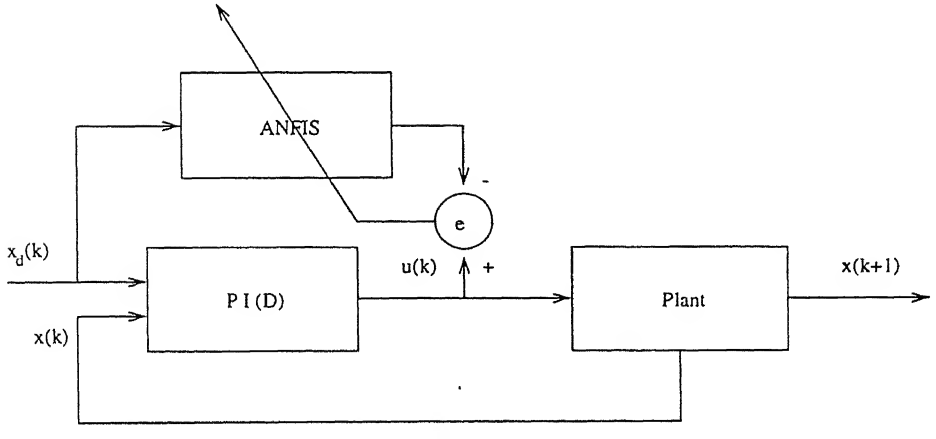


Figure 3.3: Feedback error learning

output (SISO) plant, the training dataset is obtained by generating $u(k)$ at random and observing the states and output $x(k)$ and $y(k+1) = x_1(k+1)$ respectively. The network is then made to learn the inverse model of plant by fitting in the data pairs $x_1(k+1)$, $x(k)$ and $u(k)$, and reducing the error $e_u(k)$ as shown in Figure 3.4(a). While using the network as controller, we provide it with desired output $x_{1d}(k)$ (or $y_d(k)$) and $x(k)$ (see Figure 3.4(b)). If the network has learned the inverse model accurately, it will generate $u(k)$, which, when fed to the plant, will produce an output $y(k+1)$ that is close to desired output $y_d(k+1)$. The existence of inverse model of plant is a necessary requirement for this method.

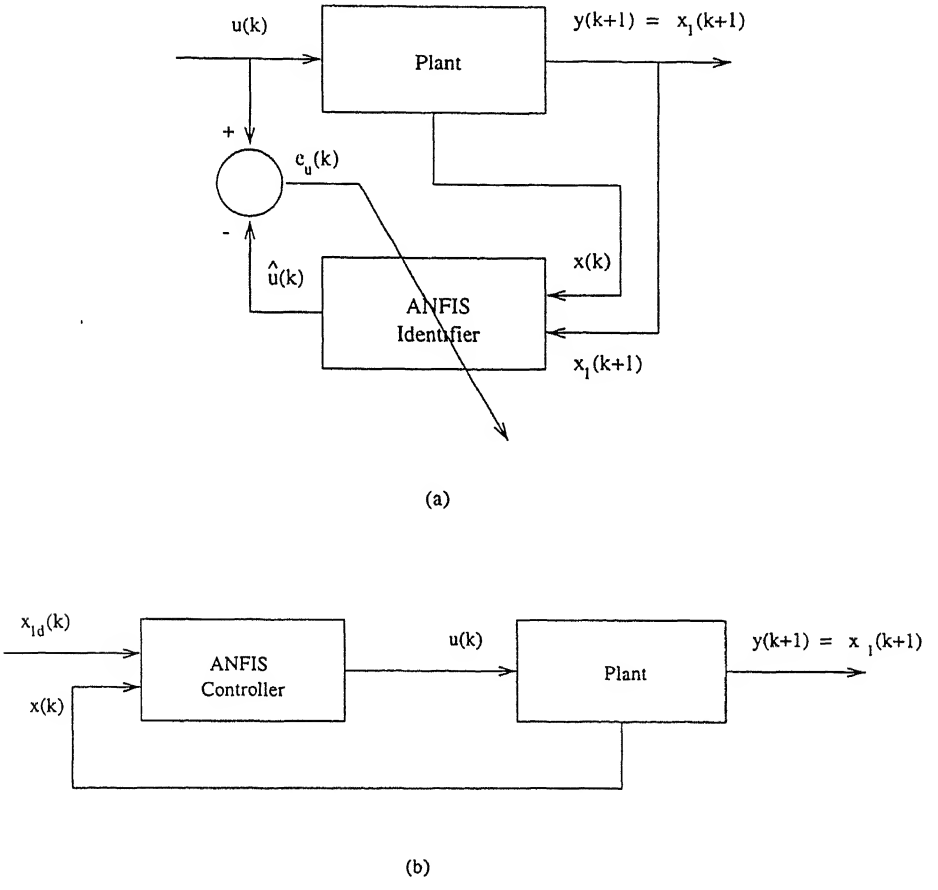


Figure 3.4: Block diagram for direct inverse control method: (a) learning phase; (b) application phase.

This method seems straightforward. However, certain problems are associated

with it. The success of this method largely depends on the ability of the network to generalize or learn to respond to situations that were not specifically used in the learning phase. This is obvious because even if we train the network for large amount of data sets, we cannot guarantee the inclusion of all possible situations that may arise in future. We may have to train the network for larger operational range than is required. This will need a lot of unnecessary training data and the training process will be time consuming. Moreover, the objective here is to minimize the network error $\| e_u \|^2$, but this does not guarantee the minimization of overall system error $\| x_d(k) - x(k) \|^2$. Therefore there is no guarantee that the output trajectory will follow the desired trajectory accurately. It has been observed that the direct inverse control method generally leads to unsatisfactory results and serious questions arise regarding its acceptability [18], [19], [21].

3.1.3 Direct Adaptive Control

Direct adaptive control or specialized learning method was proposed to overcome the problems of direct inverse control [20]. In this approach (see Figure 3.5), the desired output trajectory of the plant $x_{1d}(k)$ is given as input to the ANFIS controller

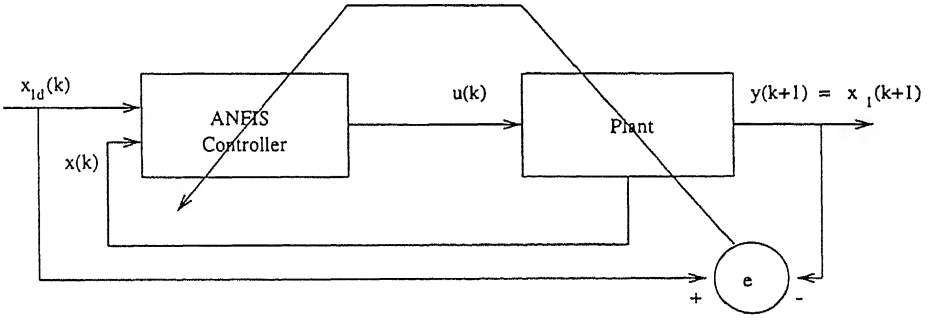


Figure 3.5. Direct adaptive control or specialized learning

which generates $u(k)$ that is fed to the plant. The plant then evolves into the next state $x(k+1)$. Controller parameters are trained using the error between actual and desired output trajectory (see Figure 3 5). Since the desired output for controller (say $u_d(k)$, is not known *a priori* we cannot apply gradient descent learning in its form described earlier (Chapter 2). A solution to this problem is to

consider the dynamic plant as an additional unmodifiable layer and propagate the error $\| x_1(k) - x_{1d}(k) \|$ through the plant. This requires the knowledge of plant derivatives or jacobian and can be achieved as shown below.

Referring to equation (2.11), the error rate of ith node in the final layer l is given by

$$\frac{\partial E_p}{\partial O^l_{i,p}} = -2(T_{i,p} - O^l_{i,p})$$

where $O^l_{i,p}$ and $T_{i,p}$ respectively are the actual and desired outputs of ith node in the final layer l. Since we do not know the value of $T_{i,p}$ (desired control input to the plant) we can modify the above equation as

$$\frac{\partial E_p}{\partial O^l_{i,p}} = -2(y_d(k+1) - y(k+1)) \frac{\partial y(k+1)}{\partial u(k+1)} \tag{3.2}$$

To use the above equation, we should have the knowledge of output sensitivity with respect to the control input u. If this knowledge is not available it can be estimated on line from the changes of plant input and output during two consecutive time instants

$$\frac{\partial y(k+1)}{\partial u(k+1)} \approx \frac{y(k+1) - y(k)}{u(k+1) - u(k)} \tag{3.3}$$

Through equation (3.2) we try to minimize an oscillating error rate and this may cause convergence problems. To avoid this problem Saerens et al [22] have suggested that $\partial y(k+1)/\partial u(k+1)$ be replaced by its sign and so equation (3.2) becomes

$$\frac{\partial E_p}{\partial O^l_{i,p}} = -2(y_d(k+1) - y(k+1)) \text{ sign} \left[\frac{\partial y(k+1)}{\partial u(k+1)}\right] \tag{3.4}$$

The information of sign$(\partial y(k+1)/\partial u(k+1))$ can be known from some prior knowledge about the plant. With the above changes, the algorithm described in Chapter 2 can be used for training the adaptive network with direct adaptive control approach.

Though this method alleviates the problems posed by direct inverse control, it has its own limitations. Since we are training the network by error minimization for a particular desired trajectory, the network learns to respond in that region only The performance of the controller deteriorates if the desired trajectory is changed considerably during the application phase We are thus able to train the controller only in certain region of interest [20]. Moreover, if we train the network on-line without any prior (off-line) training, the controlled system may become unstable [19]. Therefore, it is necessary to prepare the initial values of the controller parameters by prior off-line training in order to avoid this problem.

3.1.4 Indirect Adaptive Control

For applying direct adaptive control method, the knowledge of plant derivatives or their signs are required, and they may not be available in general. In the indirect adaptive control scheme [23], [24] (shown in Figure 3.6), before training the adaptive network controller, a separate network (known as the plant emulator) is trained

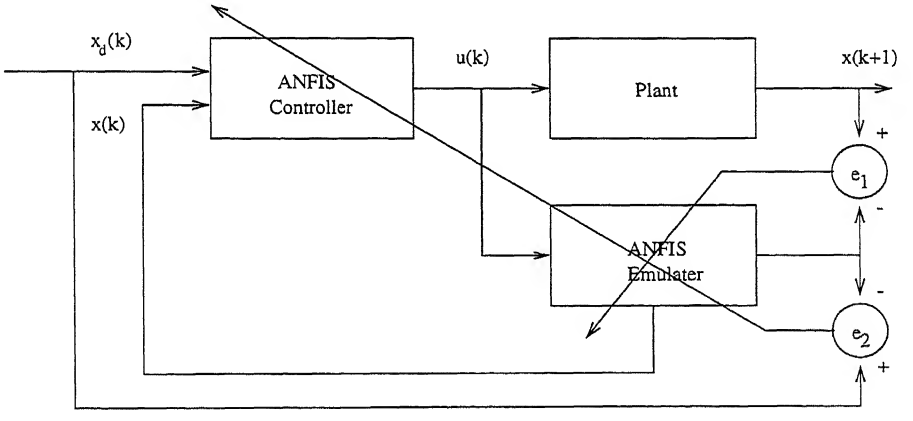


Figure 3.6: Indirect adaptive control

to behave like the plant. This emulator is then used to compute the plant output sensitivity with respect to the controller output by propagating the error measure e_2 through the emulator (see Figure 3.6). The emulator should be trained off-line with a sufficiently rich data set to allow plant identification, and then both controller and emulator are trained on-line in the application phase. This approach is advantageous

is advantageous for plants with time varying characterstics, as the emulator will be able to capture the system characterstics and accordingly the controller parameters can be tuned.

In this case however, the quality of trained emulator is crucial to the controller's performance. If the controller is operated in a range for which the emulator was not trained, the back propagation through emulator fails causing poor or even unstable control performance.

3.1.5 Model Reference Control

In model reference control [18], the performance of closed loop system is specified through a stable reference model M, which is defined by its input output pair $\{r(k), y^r(k)\}$, where $y^r(k)$ is the desired output of the plant. The control system attempts to determine the control input $u(k)$ so that the plant output $y^p(k)$ matches the reference model output asymptotically i.e.

$$\lim_{k\to\infty} \| y^p(k) - y^r(k) \| \leq \epsilon$$

for some specified constant $\epsilon \geq 0$

In this approach, indicated in Figure 3.7, the error defined above is used to train the adaptive network controller.

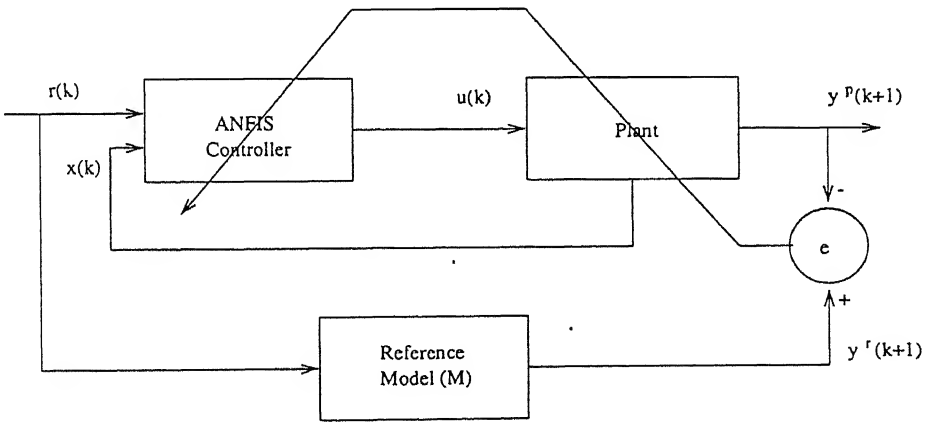


Figure 3 7 Model reference control

3.2 Preliminary Simulation Results

In this thesis, attention was focussed on direct inverse and direct adaptive control methods. To observe the performance of these approaches, some preliminary simulations were done on a linear system . The results obtained are mentioned in the following part.

3.2.1 Results With Direct Inverse Control

The linear system taken for simulations is given by:

$$\left.\begin{array}{rcl} \dot{x} & = & Ax + Bu, \\ y & = & Cx \end{array}\right\} \tag{3.5}$$

here,

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -8 & -12 & -5 \end{bmatrix}; \; B = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}; \; C = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$$

As the system is in phase variable canonical form, it is controllable. With a sampling time of 10 ms, 500 samples of $u(k)$ were generated randomly between [-1,+1] and the values of $x_1(k+1)$, $x_1(k)$, $x_2(k)$, $x_3(k)$ were stored. Each input was associated with two membership functions, giving rise to a total of sixteen rules. The initial MFs were formed so as to span the input space equally along the operating range of each input variable. The initial control action was kept zero by setting all the consequent parameters to zero.

The network was then trained by fitting the above data pairs and trying to minimize the error between the actual control variable $u(k)$ and estimated value $\hat{u}(k)$. While training, the initial step-size k was varied adaptively depending on the value of error measure. After 500 epochs of training, the error did not converge in the above case We tried different combinations by changing the number of rules from 16 to 81 and to 216, also we changed the shape of initial membership functions by varying the premise parameters, but there was no improvement in the convergence

of error. One of the possible cause of this can be weak correlation between input $u(k)$ and corresponding plant output $y(k+1)$ as shown in Figure 3.8. We can see that even though the plant input is varied between [-1,+1], there is no significant

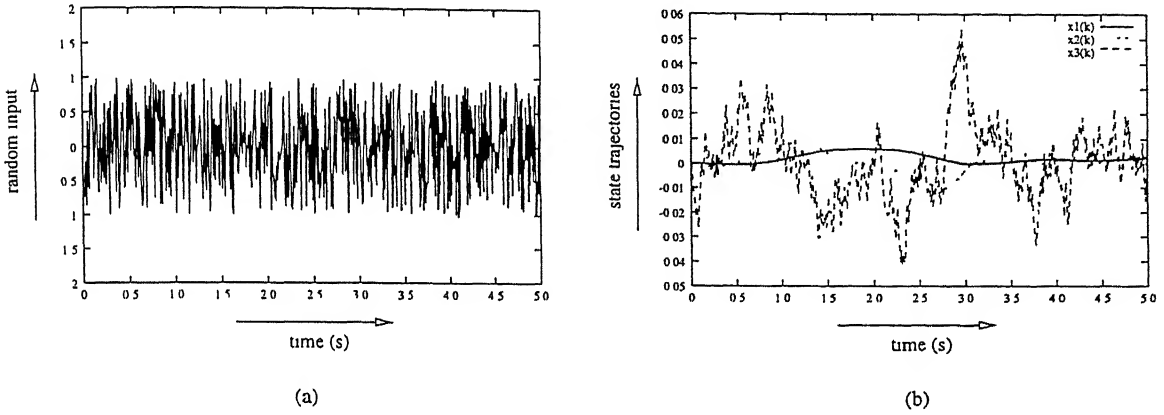


Figure 3.8: (a) random input given to the linear system; (b) resulting plant trajectories

effect on the output. The magnitude of input variation was increased to [-3,+3], still no considerable improvement was achieved.

Since the plant output was not able to respond to such fast variations in the input signal, we changed the control input to a signal as shown in Figure 3.9. The number of rules chosen for this case were sixteen, and the network parameters

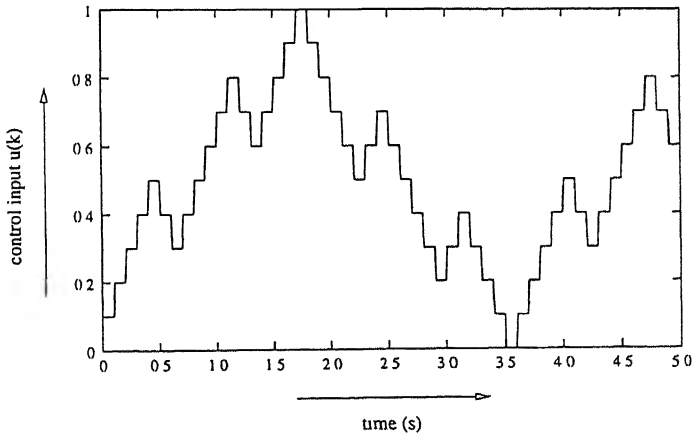


Figure 3.9: Changed control input u(k)

were initialized in the same manner as mentioned earlier. With an initial step-size

of 0.1, the network was trained for 50 cycles. Finally, the root mean square error (RMSE) between actual and desired control input was reduced to 0.0305. Desired and actual network outputs on training are shown in Figure 3.10(a) and desired and actual plant trajectories are indicated in Figure 3 10(b). It can be seen that although the actual control input matches closely to the desired value, there is a

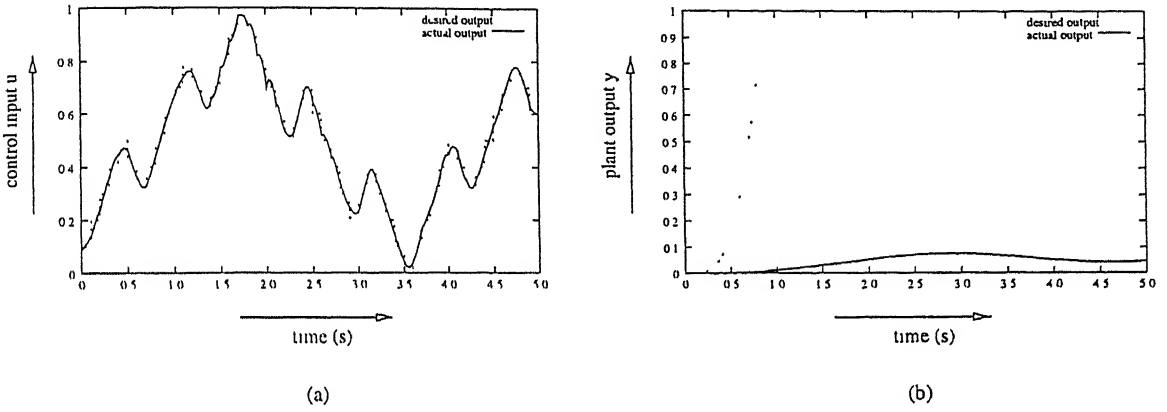


Figure 3.10: (a) control input u(k) and network output $\hat{u}(k)$; (b) desired and actual plant trajectories

considerable difference between the actual and desired plant trajectories. This is an inherent problem of direct inverse control, which is reflected in the above results and can be owed to the fact that while training we are minimizing the network error $\| u(k) - \hat{u}(k) \|$ and not the overall system error $\| y_d(k) - y(k) \|$.

3.2.2 Results With Direct Adaptive Control

Same linear system, mentioned in the previous section, was used for the simulation studies for this case also. The controller was trained off-line to track a desired trajectory having 500 samples with a sampling time of 100 ms (Figure 3.11(a)). A network having sixteen rules (two MFs per input) was used for the above purpose. The initial parameters were generated in the same way as mentioned earlier and step-size k was taken as 0.1. Initially it was observed that the network was unable to train itself in the above situation. A thorough observation revealed that although the gradient descent part was working fine in the hybrid algorithm, the least squares

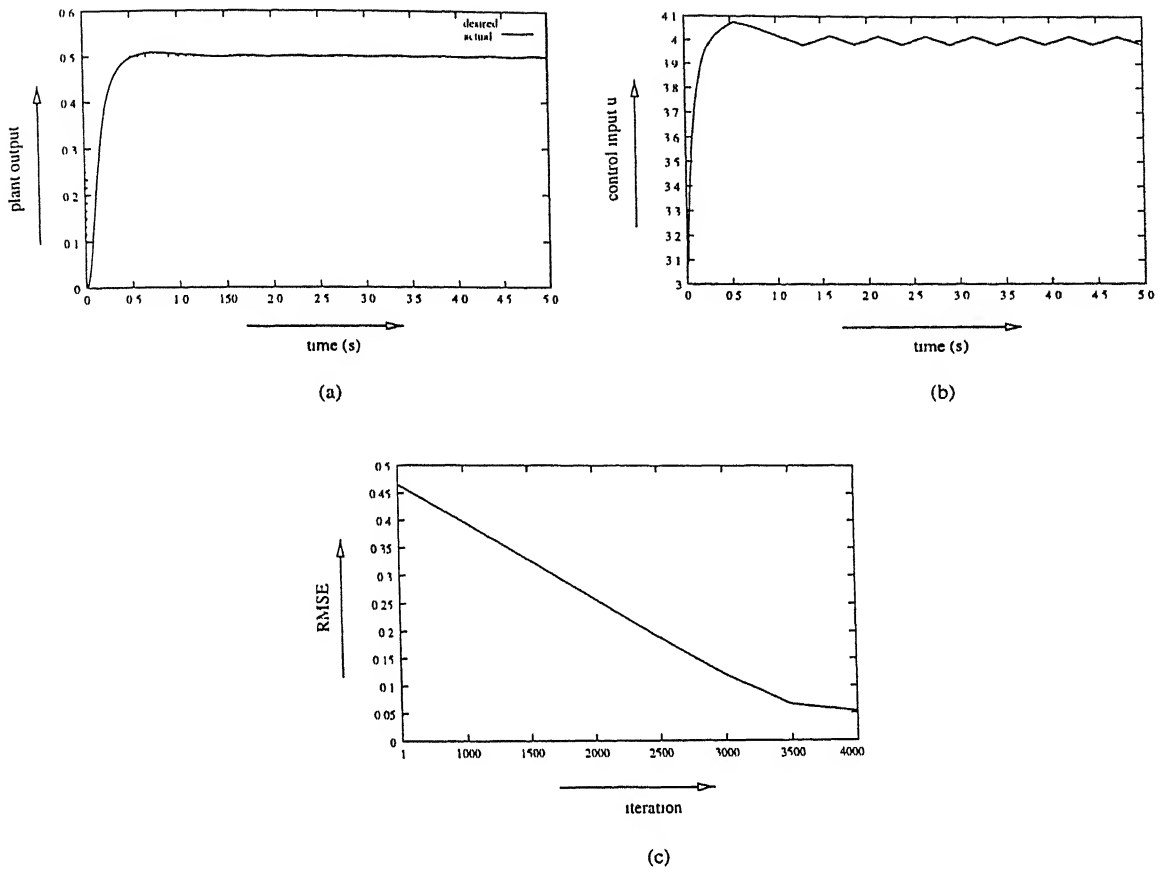
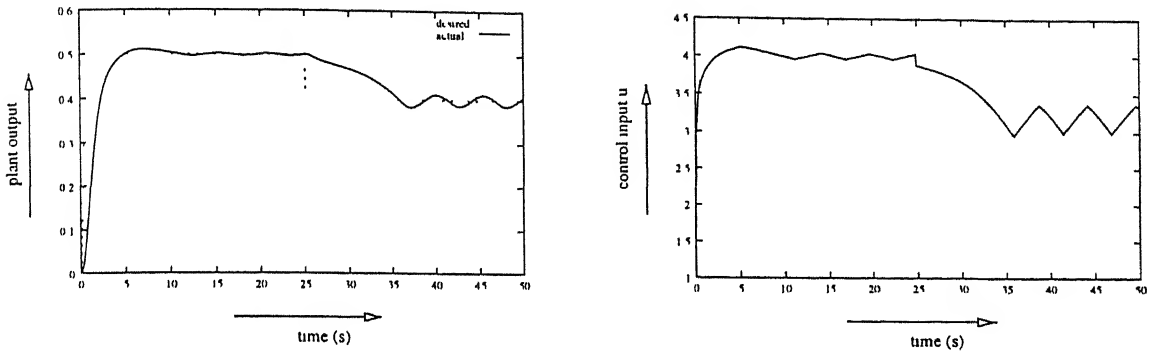


Figure 3.11: (a) desired and actual plant outputs while training; (b) control input u(k); (c) RMSE training error

estimation (LSE) was not able to identify the consequent parameters correctly. The cause of this was that we were not able to provide the desired input-output data pairs (i.e. controller inputs and output) to the least squares algorithm. This is due to the fact that it is not possible to know *a priori* which control input will take the plant output to the desired trajectory. This problem did not come in the gradient descent part because of the use of plant output sensitivity with respect to the control input $u(k)$.

For the solution of above problem, the LSE part was removed, and all the parameters in the network were updated using gradient descent. Step-size was reduced to 0.001, since large oscillations were observed in the plant output for $k = 0.1$. The results on training are shown in Figure 3.11. For testing, the desired

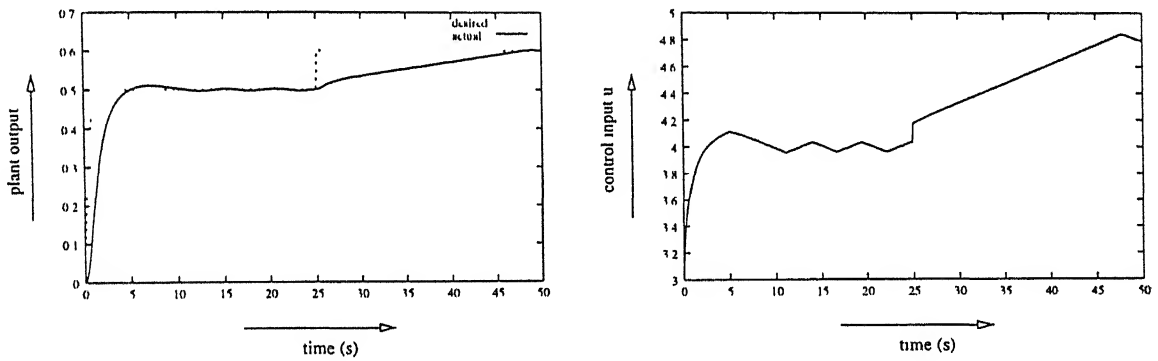
output was modified and is shown in Figure 3.12(a). Step-size was increased to 0.05 during on-line application (low value of k gave poor performance) of this trained



(a) (b)

Figure 3.12: (a) desired and actual plant outputs on testing; (b) control input u(k)

controller. The results are shown in Figure 3.12 The results of testing with another trajectory are indicated in Figure 3.13. From the above we can see that the controller



(a) (b)

Figure 3.13: (a) desired and actual plant outputs on testing; (b) control input u(k)

performance deteriorates if the testing trajectory is different from the one on which network was trained. This is a major drawback of direct adaptive control.

Chapter 4

Speed Control of a Permanent Magnet DC Motor

In the most common applications of a dc drive system (in rolling mills, paper mills etc.), the speed of the drive is required to vary along its range of operation. This variation is brought about by mostly controlling the armature voltage of the drive, which is fed through a six or twelve pulse controlled rectifier. Accurate speed tracking becomes an important factor for any high performance drive system. The direct adaptive control method, explained earlier, can be used for the above purpose if the desired speed trajectory is known *a priori*. Small variations around this trajectory can be tracked on-line. This chapter describes the ANFIS controller application to a permanent magnet dc (pmdc) motor.

4.1 PMDC Motor Model

The pmdc motor model is indicated in Figure 4.1. The outer speed loop consists of ANFIS controller, which is used for speed regulation of the motor, and the inner current loop consists of a proportional controller to limit the motor current. Motor drives a quadratic load which makes the overall system nonlinear in nature. The system dynamics are defined by the following equations [25]

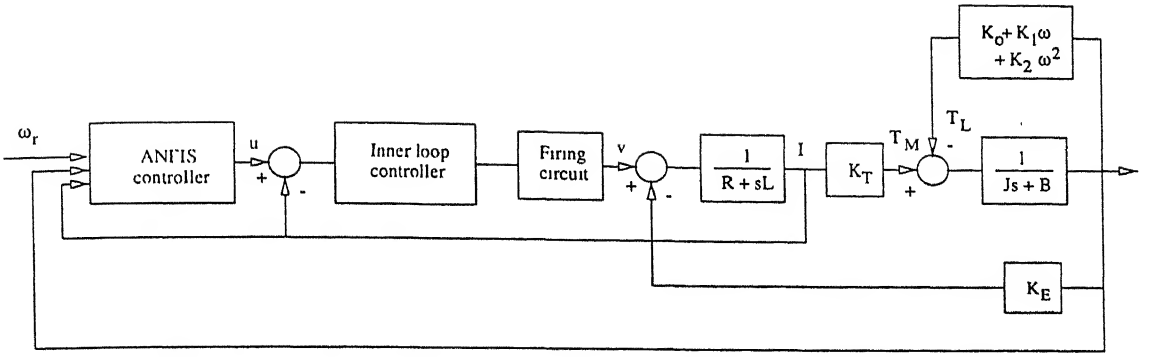


Figure 4.1: Block diagram of the pmdc drive system

Load equations:

$$T_L = K_0 + K_1\omega + K_2\omega^2 \tag{4.1}$$

$$T_M = K_T I \tag{4.2}$$

$$T_M - T_L = J\dot{\omega} + B\omega \tag{4.3}$$

where ω is the speed of the motor, T_M is the motor torque, T_L is load torque, I is the motor current, J and B are motor inertia and damping constants, respectively, and K_T is motor torque constant.

Electrical equations:

$$E = K_E\omega \tag{4.4}$$

$$V - E = RI + L\dot{I} \tag{4.5}$$

where E is the motor emf, K_E is the back emf constant, L and R are the resistance and inductance of motor armature circuit and V is the armature voltage (control input). V is given by

$$V = 1.35V_{LL}Cos\alpha \tag{4.6}$$

where α is the firing angle for a six-pulse rectifier bridge. The rectifier firing angle is limited between 5° and 89°. Combining equations (4 1) to (4.6) and taking motor speed and current as state variables, we get the following two first order state equations:

$$\left.\begin{aligned} \dot{\omega} &= \frac{-(B+K_1)}{J}\omega + \frac{K_T}{J} I - \frac{K_2}{J}\omega^2 - \frac{K_0}{J} \\ \dot{I} &= \frac{-K_E}{L}\omega - \frac{R}{L} I + \frac{V}{L} \end{aligned}\right\} \tag{4.7}$$

As mentioned earlier, the control structure of the pmdc motor system consists of an inner current loop which is to limit the motor armature current. A proportional controller is chosen for this which is of the form

$$V = 1.35 V_{LL} C_p (u - I) \tag{4.8}$$

where C_p is the proportional gain and u is the output of speed regulator. Combining equations (4.7) and (4.8), we get

$$\left.\begin{aligned} \dot{\omega} &= \frac{-(B+K_1)}{J}\omega + \frac{K_T}{J} I - \frac{K_2}{J}\omega^2 - \frac{K_0}{J} \\ \dot{I} &= \frac{-K_E}{L}\omega - \frac{(R+1.35 V_{LL} C_p)}{L} I + \frac{1.35 V_{LL} C_p u}{L} \end{aligned}\right\} \tag{4.9}$$

which gives the reduced system shown in Figure 4.2.

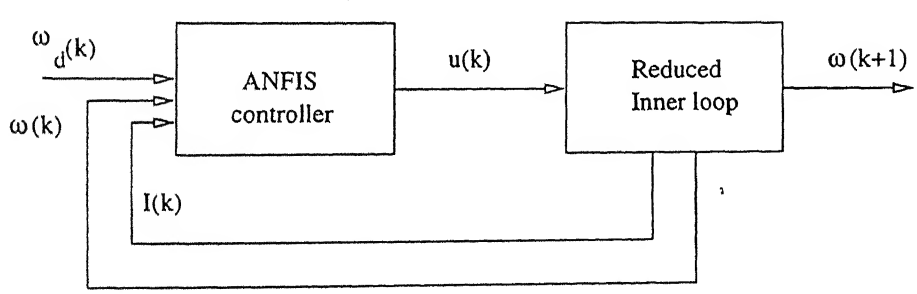


Figure 4.2: Reduced block diagram of the pmdc drive system

4.2 Tracking a Known Trajectory

When the desired speed trajectory is known *a priori*, we can train the controller off-line on this trajectory and then this tuned controller can be applied in the on-line phase for giving desired performance. The pmdc motor system is simulated for tracking two different speed trajectories. A 240[V], 2[HP], 600[rpm] motor is selected for above purpose, whose parameter values in per unit are given in the Appendix. The proportional controller gain is taken as 0.80 [25]. A sampling time of $\Delta T = (10/3)$ms is taken for this simulation, which corresponds to the time between two successive firing instants in a six-pulse rectifier.

A network with eight rules (two MFs per input) is used, with the premise parameters initialized to equally span the input space. The consequent parameters are all set to zero to give a zero initial control action. A step-size of 0.0001 is chosen for tracking a desired output shown in Figure 4.3(a). The results of the controller performance are given in Figure 4.3. We can see that the system has been able to track the specified trajectory satisfactorily. Motor current is indicated by Figure 4.3(b). The choice of $C_p = 0.8$ has limited the armature current between 0.0 and 1.4 p.u. Figure 4.3(c) shows the rectifier firing angle. Spikes in firing angle indicate the instants at which speed was changed suddenly. The root mean square error (RMSE) between the desired and actual speeds is given by Figure 4.3(d). It can be seen that as training progresses (number of epochs increase), there is a smooth reduction in the RMSE. Training was discontinued after the RMSE became sufficiently constant. Figure 4.4 shows the initial and final (tuned) membership functions.

The controller is also trained for a separate trajectory, Figure 4.5(a), and the results are indicated in Figure 4.5. The initial and final MFs are shown in Figure 4.6. A significant observation in our simulations is that low value of step-size k gives a damped response, whereas for large value of k the system response becomes oscillatory in nature

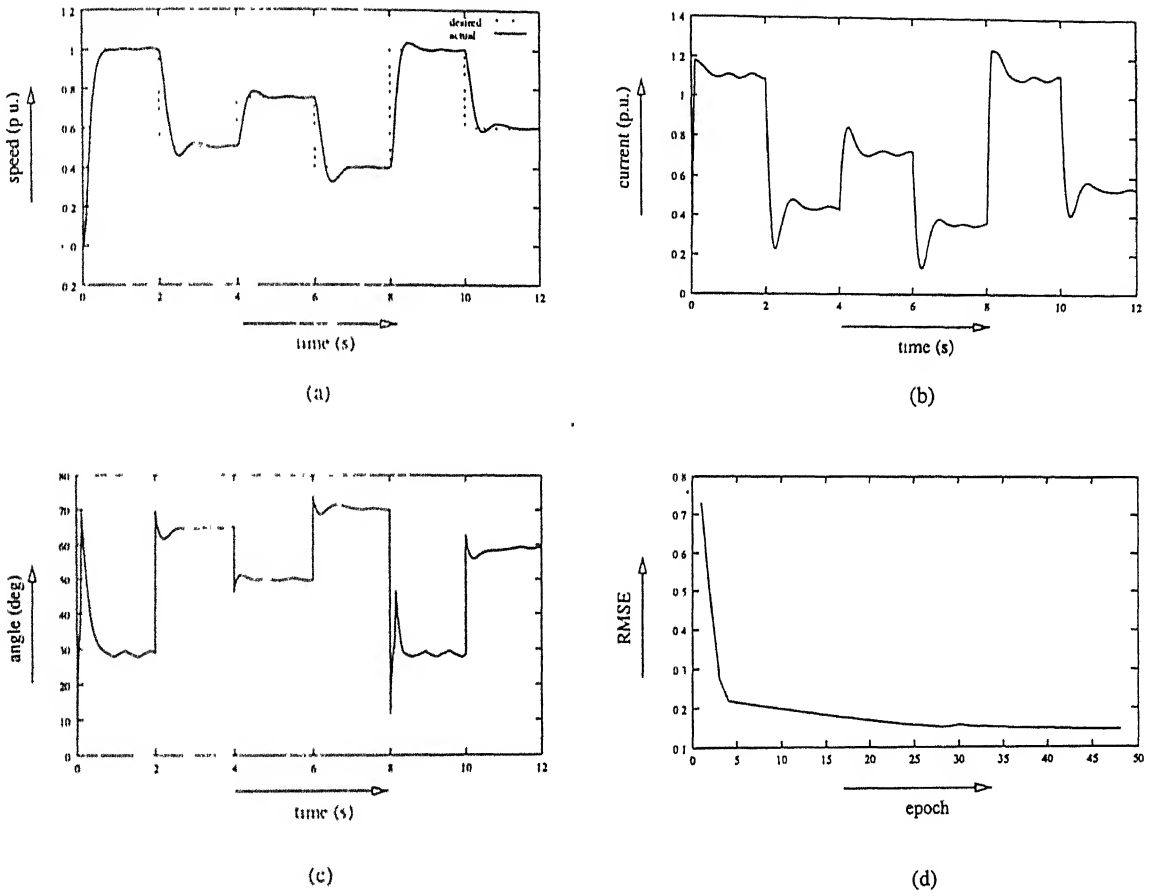


Figure 4.3: Desired speed tracking : (a) desired and actual trajectories; (b) motor armature current I; (c) rectifier firing angle α (d) RMSE training error.

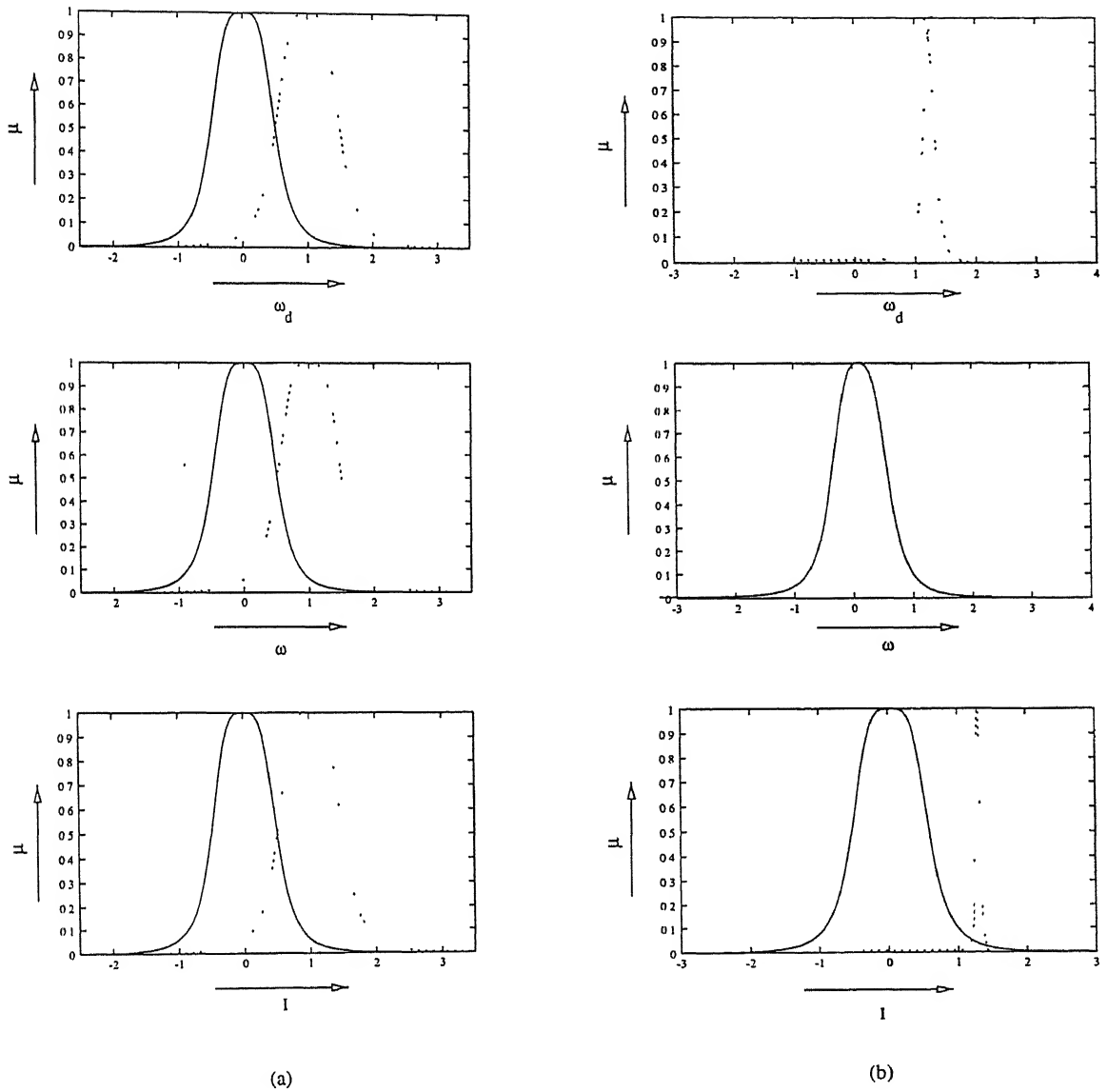


Figure 4 4 Membership functions for the results of Figure 4.3: (a) initial; (b) final.

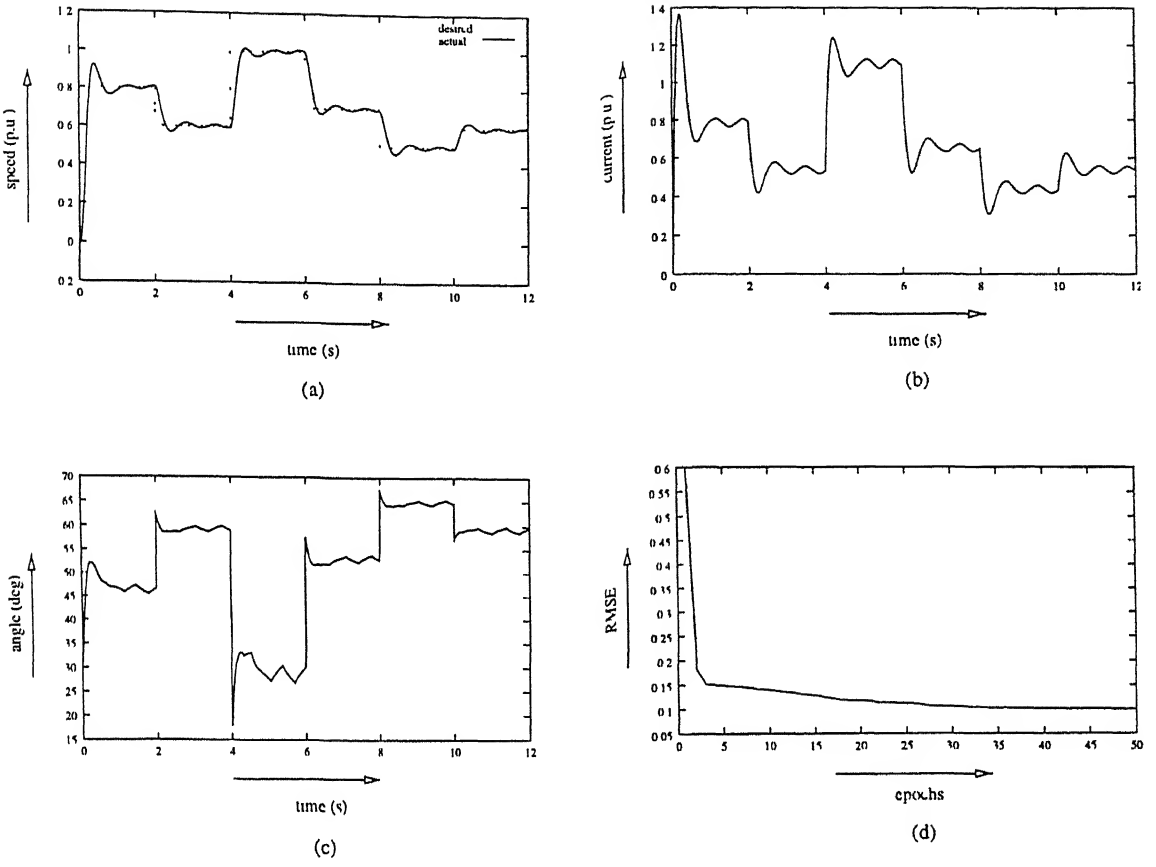


Figure 4.5: Performance on a different trajectory : (a) desired and actual speeds; (b) motor armature current I; (c) rectifier firing angle α (d) RMSE training error.

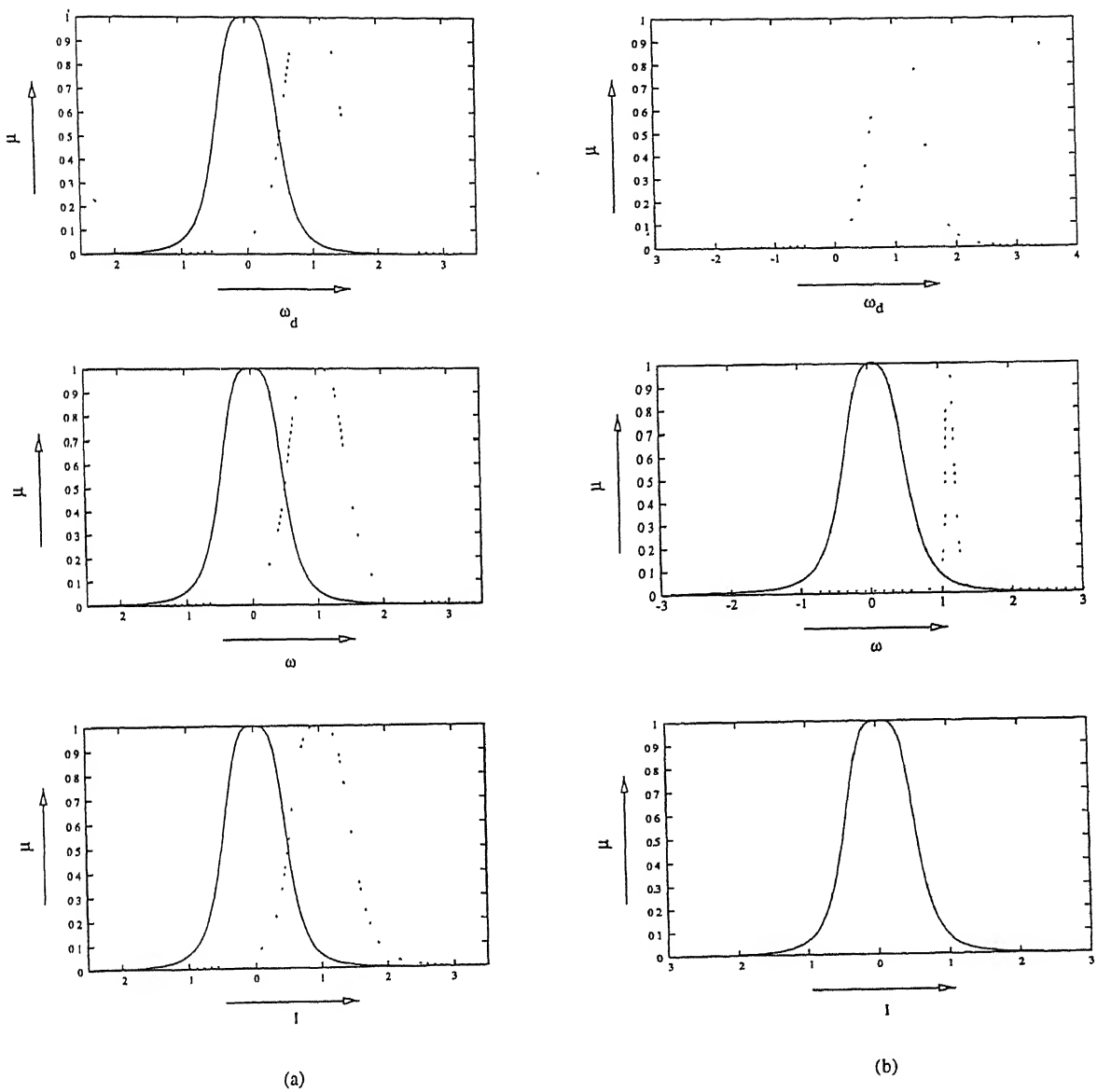


Figure 4 6: Membership functions for the results of Figure 4.5. (a) initial, (b) final.

Chapter 5

Disturbance Rejection in a Buck-Boost Converter System

The buck-boost converter system is chosen as a second example for observing the performance of ANFIS controller. This system consists of a linear circuit that is switched between two configurations and hence the overall behaviour of the system becomes nonlinear. In this, the final state of one configuration becomes the initial state of the following configuration and vice versa, and in this way the cycle repeats. ANFIS controller, in the direct adaptive control mode, is applied to the exact model of the system for maintaining its output voltage constant when there is some random disturbance present in the input dc voltage.

5.1 System Model

The circuit for a buck-boost converter is as shown in Figure 5.1, in which the converter is supplied by a dc source. The output of the circuit is indicated by V_o with polarity as shown in Figure 5.1. This circuit is switched periodically with a time period T. The switch S remains closed for a time dT and open for $(1-d)T$ in each cycle. Here, d $(0 \leq d \leq 1)$ is called as the duty ratio of switching, which regulates the average output voltage $(\overline{V_o})$ of the system. The relation between d and $\overline{V_o}$ is given by

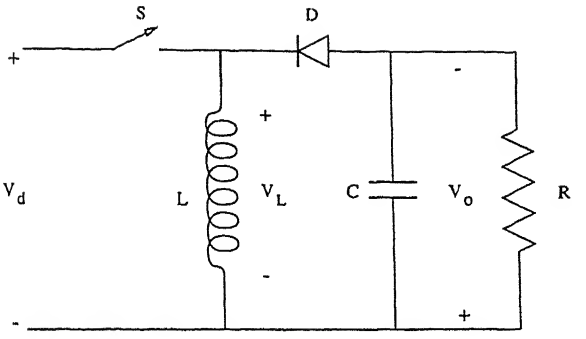


Figure 5 1: Buck-boost converter: circuit details

$$\overline{V_o} = \frac{d}{1-d} V_d \tag{5.1}$$

where V_d is the dc input to the system.

When the switch S is closed (system in mode 1), the inductor is connected to the input dc voltage V_d, and R & C are connected in parallel (see Figure 5.2(a)). In this case, the input supplies energy to the inductor, and diode D is reverse biased.

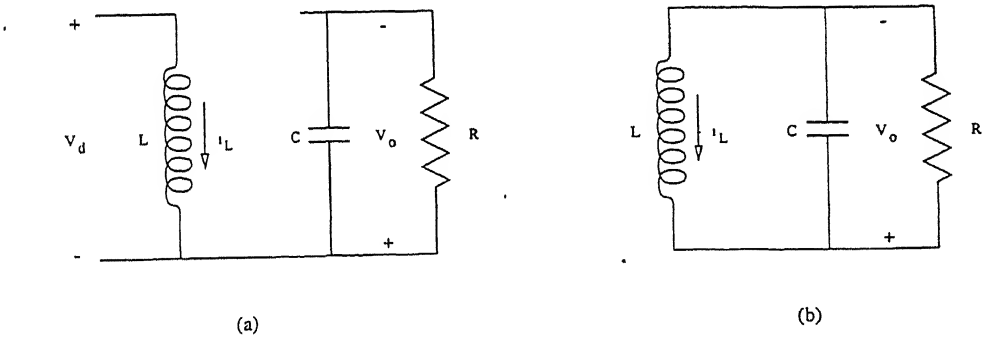


Figure 5 2: Buck-boost converter in: (a) switch on mode; (b) switch off mode.

When the switch is open (system in mode 2), the inductor, capacitor and resistance are connected in parallel (diode D forward biased), and the energy stored in the inductor is transferred to the output (Figure 5.2(b)). The system equations in these two modes of operation are given by [26]

Mode 1 $(t_\circ \leq t \leq t_1,\ t_1 = dT)$:

$$\dot{x} = A_1 x + B_1 V_d \tag{5.2}$$

Mode 2 $(t_1 \leq t \leq t_2,\ t_2 = (1-d)T)$:

$$\dot{x} = A_2 x + B_2 V_d \tag{5.3}$$

and the output is given by

$$y = Cx = V_\circ \tag{5.4}$$

Defining a state vector $x = [v_C\ i_L]^T$, the matrices in equations (5.2) - (5.4) are given by

$$A_1 = \begin{bmatrix} -1/RC & 0 \\ 0 & 0 \end{bmatrix};\ A_2 = \begin{bmatrix} -1/RC & 1/C \\ -1/L & 0 \end{bmatrix};$$

$$B_1 = \begin{bmatrix} 0 \\ 1/L \end{bmatrix};\ B_2 = \begin{bmatrix} 0 \\ 0 \end{bmatrix};\ C = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

5.2 Disturbance Rejection Using ANFIS

ANFIS controller was applied to the buck-boost converter model mentioned above, for keeping its output average voltage constant when some random disturbance (noise) was present in the input dc voltage. For this purpose, the following values were chosen as the system parameters

$$L = 4 \text{ mH};\ C = 25\mu \text{ F};\ R = 10\Omega;\ V_d = 100 \text{ V}$$

These values gave positive inductor current even for a duty ratio of 0.05, and hence the system operated only in continuous mode of conduction. The switching frequency of the circuit was chosen to be 1500 Hz. A block diagram of the converter system along with controller is shown in Figure 5.3. The dc voltage V_d (containing

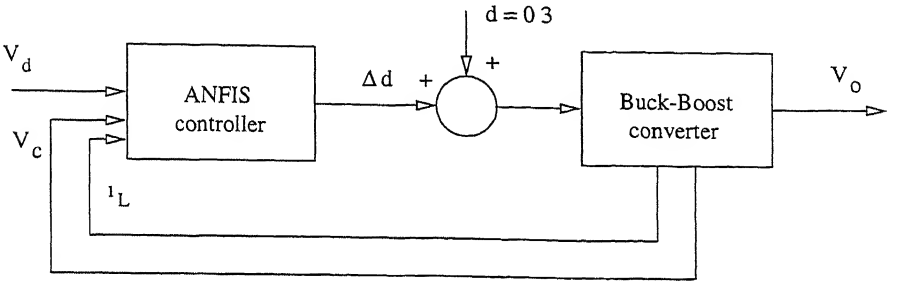


Figure 5.3 Block diagram of a buck-boost converter control system

noise), and state variables, capacitor voltage and inductor current, were given as inputs to the network. The corresponding controller output ($\triangle d$), was fed to the plant, in addition to a fixed value of $d_o = 0.3$ The objective of the combined system is to maintain a constant output voltage corresponding to this value of d_o and is given by

$$\overline{V_o} = \frac{d_o}{1-d_o} V_d = \frac{0.3}{1-0.3} 100 = 42.86V$$

The nominal states for $d_o = 0\ 3$ are given by

$$x_o = [43.46\ 2.87]^T$$

A network with eight rules (two MFs per input) was formed and its parameters initialized. Sampling time of $\triangle T = 1/150000$ was taken for simulating this system. The step-size was fixed to 0.0001. The input voltage V_d was generated by adding a random disturbance between [-10,+10] V to a value of 100 V. Also, for some time the input was kept noise free (see Figure 5.4(a)). The results of the simulation are shown in Figure 5.4 and initial and final membership functions are shown in Figure 5.5. We can see that when the disturbance is present in the input voltage V_d, the controller output $\triangle d$ varies to change the switching instant in a cycle, and hence controls the average output voltage. On removing the disturbance, $\triangle d$ becomes constant, to

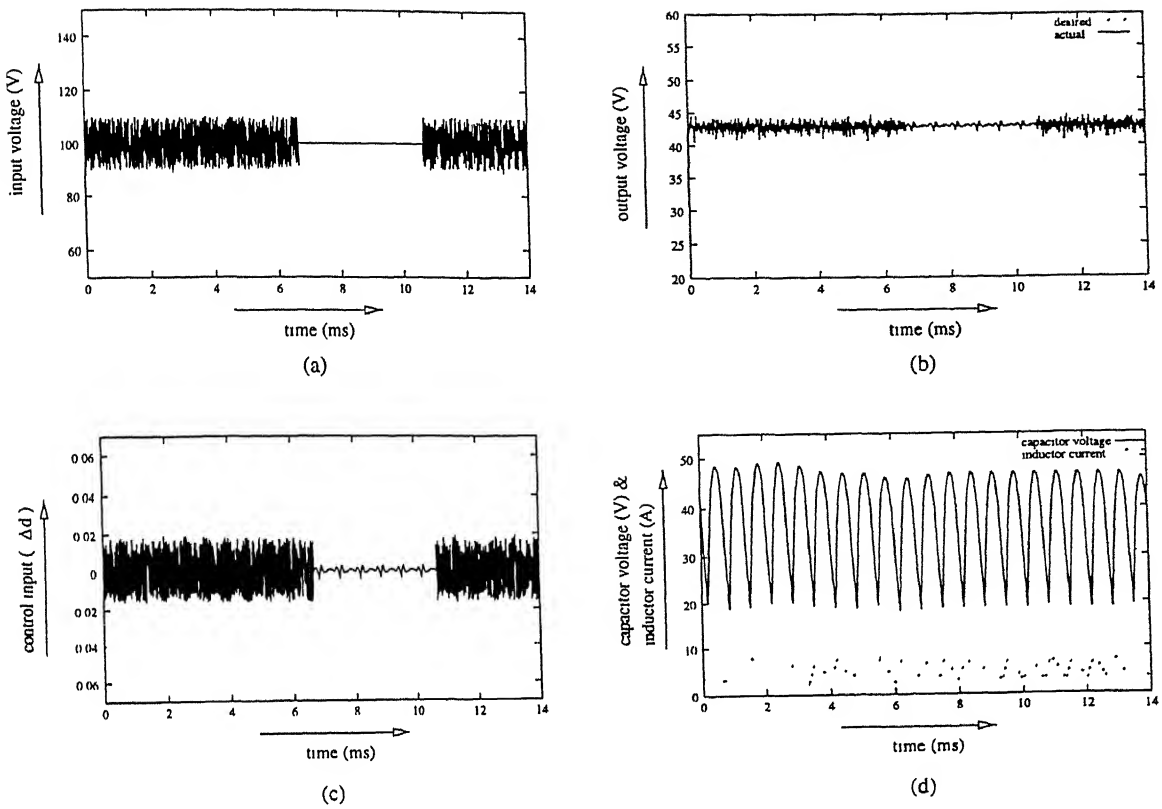


Figure 5.4: Simulation results for a network with three inputs and eight rules: (a) input dc voltage V_d; (b) desired and actual output voltages; (c) control input Δd; (d) capacitor voltage and inductor current.

give constant average output voltage. However, small oscillations are visible in the value of Δd and can be owed to the fact that the feedback to the controller consists of capacitor voltage and inductor current, which contain ripples. On removing this feedback, and using four rules, Δd became perfectly constant (Figure 5.6). The average output voltage is shown in Figure 5.6(b). It shows that although the system has not been able to eliminate the noise completely, its magnitude is reduced to [-1.5,+1.5]. Figure 5 7 shows the initial and final membership functions.

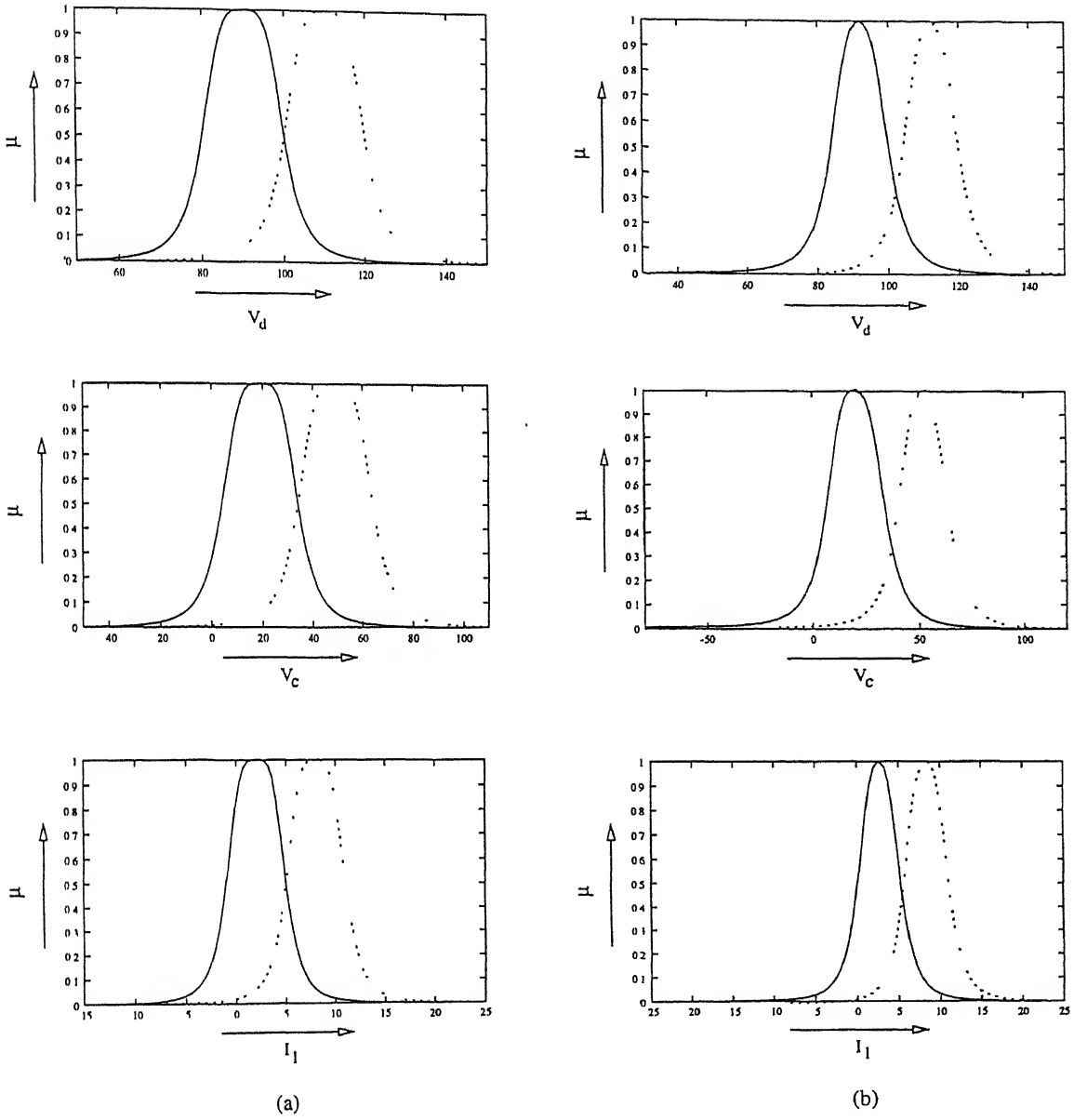


Figure 5 5: Membership functions for the results of Figure 5.4· (a) initial; (b) final.

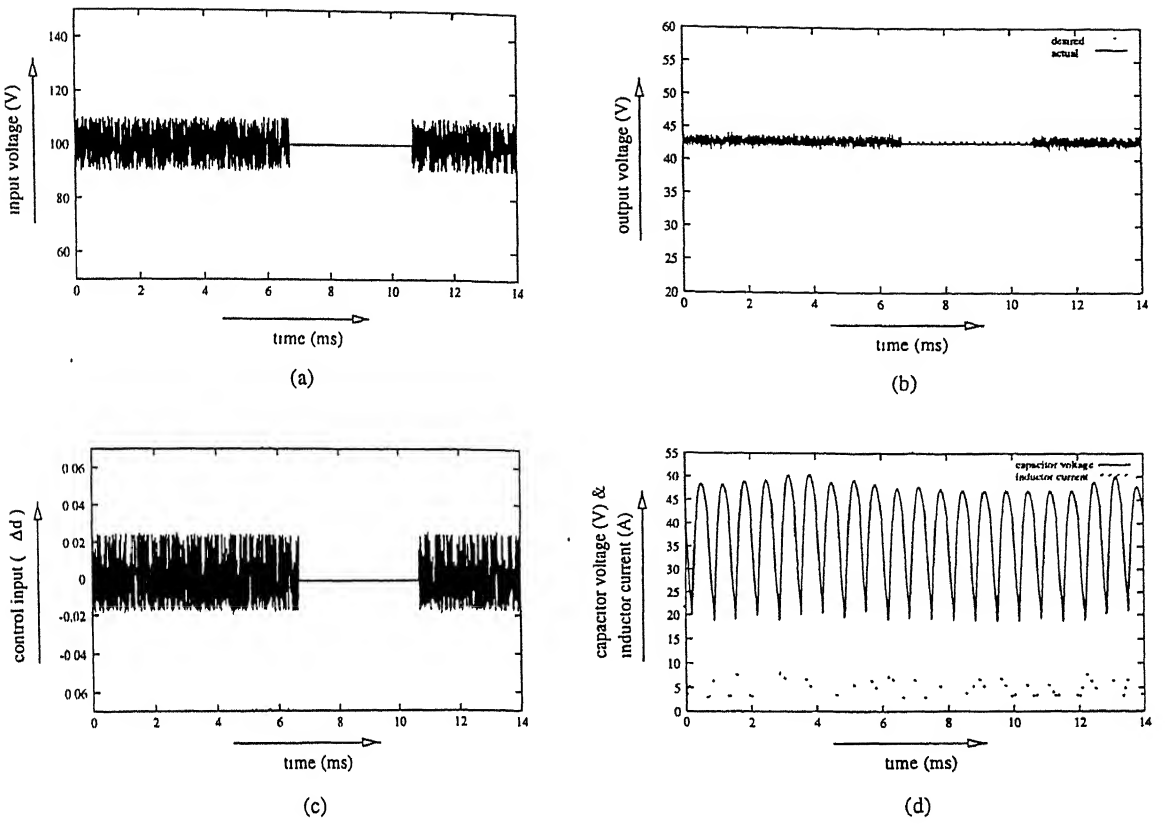


Figure 5.6: Simulation results for a network with one input and four rules: (a) input dc voltage V_d; (b) desired and actual output voltages, (c) control input Δd; (d) capacitor voltage and inductor current.

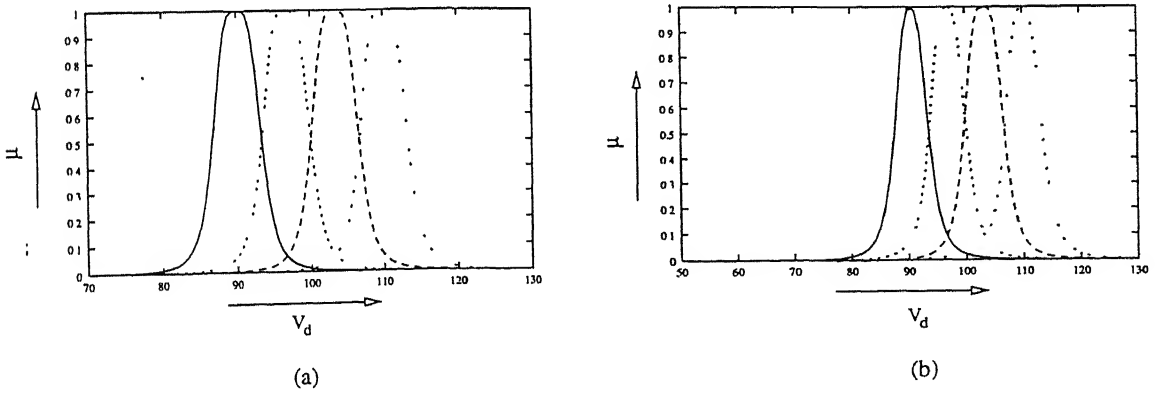


Figure 5 7. Membership functions for the results of Figure 5.6: (a) initial; (b) final.

Chapter 6

Conclusions

In this thesis, an attempt has been made to evaluate the performance of a class of neuro-fuzzy controllers. Such controllers come into picture when the classical control techniques fail in generating a satisfactory solution, which is generally the case for nonlinear/ ill-defined systems. The neuro-fuzzy system used here is formed by converting the Sugeno fuzzy model into an adaptive network and tuning its parameters using gradient descent or hybrid learning algorithm discussed earlier. Some of the conclusions and scope for future work is given in this Chapter.

6.1 General Conclusions

In the present work, we have focussed our attention on two control structures: direct inverse control and direct adaptive control. These can be derived using ANFIS. Preliminary simulations were done on a simple linear system using these structures and some important points were observed. The direct inverse control scheme, when applied on the specified system, was found to be unsuitable for tracking the desired plant output. Different combinations were tried on this method by changing the number of MFs per input (number of rules), number of training data samples, stepsize and initial parameters. Once a minimum number of rules were defined, we did not obtain any significant improvement on the RMSE on increasing the number of rules. On the contrary, because of increased number of rules, the network became

more complex and training time is increased. After a minimum number of samples in the dataset (> number of modifiable parameters) were available an increase in the training data did not alter the performance of the network. The initial step-size was mostly kept around 0.10. In between it was varied to check whether the network was getting trapped into a local minima while training. Furthermore, since we were adaptively updating the stepsize, the initial value is not very critical as long as it is not too big.

Usually bell shaped initial membership functions were used. Their shapes were varied to see the convergence property of the network. This did not have any substantial effect on the network performance. Even after applying the above changes, no significant improvement was observed in our simulation results. The unsatisfactory performance of direct inverse control scheme has also been reported in the literature [18], [19], [21].

Direct adaptive control method was then applied to alleviate the above problems. On the basis of its performance on specified linear system and two other nonlinear control problems (*speed control of pmdc motor system* and *disturbance rejection in a buck-boost converter*), the following points were observed. This method performed well when the desired trajectory to be tracked was known *a priori*. Although the system was able to track some small variations around the trajectory on which training was done, but its performance deteriorated when the desired output was changed considerably in the application phase. Better performance in this respect is reported in the literature using variable gain control approach [25]. It can therefore be concluded that practical applications of this technique will be limited, as every time the the desired trajectory changes, controller will have to be re-tuned. Also, for each training cycle (epoch) we will have to operate the plant, which may be costly and time consuming. The stability performance of these systems is also a cause for concern. Though current research is going on in this area, very few results are available at present [28].

Training forms a very important part in the performance of the adaptive network. This area is primarily guided by heuristics and formal training procedures, such that the network captures the desired controller dynamics sufficiently well are currently

not available. Furthermore, results regarding the network size, number of rules etc. for optimum network performance are not present. A firm theoretical basis is required in this regard.

The control techniques discussed in this work did not give the desired performance level for their application to problems of practical importance. Certain control techniques based purely on fuzzy methodologies have been more successful than ANFIS. The reasons for this can be attributed to the lack of firm theoretical framework in the field of fuzzy-neural systems. However, since this field is still in a fluid state and a lot of research is in progress in both, the theoretical and application domain, better results can be expected in the near future.

6.2 Scope for Further Work

Although, various factors were considered while applying ANFIS controller in the above work, there are still some important areas which can form the subject of further investigation. In our simulations, network training was started from a point where it has no prior knowledge about the system. Faster convergence and better generalization may be achieved if some initial knowledge can be incorporated into the network in the form of initial parameters and number of rules etc. In direct adaptive control, this can form the part of off-line learning, while during on-line application the network can refine these parameters to give better generalization.

In this thesis, while applying the gradient descent algorithm, we have calculated the error rates based on error measure E_p. This can be changed to a suitable objective function to realize different control goals. Models other than the Sugeno fuzzy model can be investigated. Even though this model captures a piecewise linear system more accurately, there is still scope for models which capture the nonlinear dynamics better. A general purpose model, which can describe various types of nonlinearities, will be a very important development.

Appendix

The parameters for the pmdc motor chosen for simulation studies in Chapter 4 are given below. The parameters are scaled by their base values.

Rated power = 2 [HP];	Rated voltage = 240 [V];
Rated speed = 600 [rpm];	Resistance R = 0.1684 [pu];
Inductance L = 0.0814 [pu];	Moment of inertia J = 0.1985 [pu];
Damping B = 0.079 [pu],	Torque constant $K_T = 1$ [pu];
Back emf constant $K_E = 1$ [pu];	Load constant $K_0 = 0.1895$ [pu];
Load constant $K_1 = 0$ [pu];	Load constant $K_2 = 0.8313$ [pu].

References

[1] D. Driankov, H. Hellendoorn and M. Reinfrank, "An introduction to fuzzy control" (Narosa Publishing House, 1996)

[2] T. Furuhashi, S. Horikawa and Y. Uchikawa, "On the stability of fuzzy control systems using a fuzzy modeling method", *Proceedings of International Conference on Industrial electronics, Control, Instrumentation and Automation*, San Diego, Vol. 2, pp 982-985, 1992.

[3] L.X. Wang, "Adaptive fuzzy systems and control: design and stability analysis" (Prentice-Hall, NJ, USA).

[4] W.T. Miller, R Sutton and P.J Werbos, "Neural networks for control" (MIT Press, Cambridge, MA, USA, 1990)

[5] Simon Haykin, "Neural networks. A comprehensive foundation" (Macmillan, New York, NY, 1994)

[6] K S. Narendra and K. Parthasarathy, "Identification and control of dynamical systems using neural networks", *IEEE Transactions on Neural Networks*, Vol. 1, pp. 4-27, 1990.

[7] K.S. Narendra and K. Parthasarathy, "Gradient methods for optimisation of dynamical systems containing neural networks", *IEEE Transactions on Neural Networks*, Vol 2, No. 2, pp. 252-262, 1991.

[8] Bart Kosko, "Neural networks and fuzzy systems: A dynamical systems approach to machine intelligence" (Prentice-Hall International Editions, 1992).

[9] J.S.R. Jang and C.T Sun, "Functional equivalence between radial basis function networks and fuzzy inference systems", *IEEE Transactions on Neural Networks*, Vol. 4, No 1, pp. 156-158, 1993

[10] J.J. Bucley and Y. Hayashi, "Hybrid neural nets can be fuzzy controllers and fuzzy expert systems", *Fuzzy Sets and Systems*, Vol. 60, pp. 135-142, 1993.

[11] L.X. Wang and J M. Mendel, "Fuzzy basis functions, universal approximation and orthogonal least squares", *IEEE Transactions on Neural Networks*, pp. 807-814, 1992.

[12] H. Takagi, N. Suzuki, T Koda and Y. Kojima, "Neural networks designed on approximate reasoning architecture and their applications", *IEEE Transactions on Neural Networks*, Vol 3, No. 5, pp 752-760, 1992.

[13] T. Takagi and M. Sugeno, "Fuzzy identification of systems and its applications to modeling and control", *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 15, pp. 116-132, 1985.

[14] J.S.R. Jang and C.T. Sun, "Neuro-fuzzy modeling and control", *Proceedings of the IEEE*, Vol. 83, No. 3, pp. 378-406, March 1995.

[15] E.H. Mamdani and S. Assilian, "An experiment in linguistic synthesis with a fuzzy logic controller", *International Journal on Man Machine Studies*, Vol. 7, No. 1, pp. 1-13, 1975.

[16] Y. Tsukamoto, "An approach to fuzzy reasoning methods", in *Advances in Fuzzy Set Theory Applications*, M.M. Gupta, R.K. Ragade and R R. Yager, Eds. Amsterdam: North-Holland, pp. 137-149, 1979

[17] J.S.R. Jang, "ANFIS: Adaptive network-based fuzzy inference system", *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 23, No. 3, pp. 665-685, May/June 1993.

[27] J.S.R. Jang, "Self learning fuzzy controllers based on temporal back propagation", *IEEE Transactions on Neural Networks*, Vol. 3, No. 5, pp. 714-723, Sept. 1992.

[28] K S. Narendra and A.U. Levin, "Control of nonlinear dynamical systems using neural networks: Controllability and stabilization", *IEEE Transactions on Neural Networks*, Vol. 4, No. 2, pp. 192-206, March 1993.

[29] A Zhang and A.J. Morris, "Fuzzy neural networks for nonlinear systems modeling", *IEE Proceedings - Control Theory Applications*, Vol. 142, No. 6, pp. 551-561, Nov. 1995.

[30] C.C Lee, "Fuzzy logic in control systems: Fuzzy logic controller - Part 1", *IEEE Transactions on Systems, Man and Cybernetics* Vol. 20, pp. 404-418, Feb. 1990.

[31] C.C. Lee, "Fuzzy logic in control systems: Fuzzy logic controller - Part 2", *IEEE Transactions on Systems, Man and Cybernetics* Vol. 20, pp. 419-435, Feb. 1990.

[32] T. J. Ross, "Fuzzy logic with engineering applications", (McGraw-Hill, Inc., 1995).